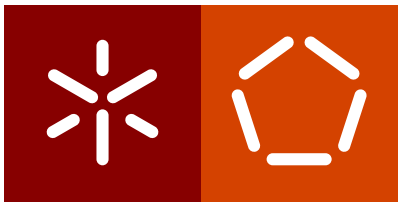
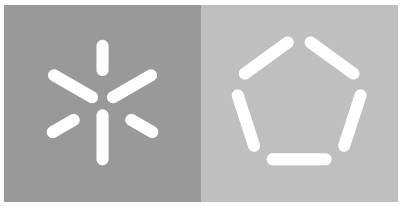


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Paulo Rafael da Costa e Sousa

An Efficient Software Tool to Segment Slice and View Electron Tomograms

April 2017



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Paulo Rafael da Costa e Sousa

An Efficient Software Tool to Segment Slice and View Electron Tomograms

Master Dissertation
Master Degree in Computer Science

Dissertation supervised by
Alberto José Proença
Daniel Grando Stroppa

April 2017

AGRADECIMENTOS

Ao meu orientador, Alberto Proença, que, com a disponibilidade, rigor e empenho característicos, me ensinou muito para além da Engenharia Informática.

Ao meu coorientador, Daniel Stroppa, pela disponibilidade e empenho, o meu especial agradecimento.

Aos professores que mais me marcaram neste meu percurso académico na Universidade do Minho: Alberto José Proença, José Bernardo Barros, António Pina e José Nuno Oliveira.

A todos os meus amigos.

Por último, mas a quem eu devo tudo:
Aos meus pais e ao meu irmão.

Obrigado!

ABSTRACT

Segmentation is a key method to extract useful information in Electron Tomography. Manual segmentation is the most commonly used method, but it is subject to user bias and the process is slow. The lack of adequate automated processes, due to the high complexity and to the low signal-to-noise ratio of these tomograms, provided the main challenges for this dissertation: to develop a software tool to efficiently handle electron tomograms, including a novel 3D segmentation algorithm.

Tomograms can be seen as a stack of 2D images; operations on tomograms usually lead to computationally intense tasks. This is due to the large amount of involved data and to the strided and random memory access patterns. These characteristics represent serious problems on novel computing systems, which rely on complex memory hierarchy architectures to hide memory access latency time.

A software tool with a user-friendly interface — *TomSeg* — was designed, implemented and tested with experimental datasets, built with sequences of Scanning Electron Microscopy images obtained using a Slice and View technique. This tool lets users align, crop, segment and export electron tomograms, using computationally efficient processes. *TomSeg* takes advantage of the most usual architectures of modern compute servers, namely based on multicore and many-core CPU devices, exploring vector and parallel programming techniques; it also explores the available GPU-devices to speedup critical code functions. Validation and performance results on a compute server are presented together with the performance improvements obtained during the implementation and test phases.

TomSeg is an open-source tool for Unix and Windows that can be easily extended with new algorithms to efficiently handle generic tomograms.

RESUMO

A segmentação é uma técnica fundamental na tomografia eletrónica para a extração de informação. A segmentação manual é o método mais utilizado, mas é um processo lento e sujeito à parcialidade humana. A falta de métodos automáticos adequados, muito devido à elevada complexidade e à baixa relação sinal-ruído destes tomogramas, conduziu aos principais desafios desta dissertação: desenvolver uma ferramenta de *software* para manusear tomogramas electrónicos de forma eficiente, que inclui um novo algoritmo de segmentação 3D.

Os tomogramas podem ser vistos como uma pilha de imagens 2D; operações sobre tomogramas costumam originar tarefas computacionalmente exigentes. Isto deve-se à grande quantidade de dados envolvidos e aos acessos espaçados e aleatórios à memória. Estas características representam problemas sérios nos mais recentes sistemas de computação, que dependem de uma complexa arquitetura hierárquica para esconder o tempo de acesso à memória.

Desenhou-se, implementou-se e testou-se uma ferramenta de *software* com uma *interface* de utilização amigável — *TomSeg* — utilizando conjuntos de dados experimentais, construídos a partir de sequências de imagens de microscopia electrónica de varrimento obtidas através de uma técnica de *Slice and View*. Esta ferramenta permite aos utilizadores alinhar, cortar, segmentar e exportar tomogramas electrónicos, utilizando processos computacionalmente eficientes. O *TomSeg* tira vantagem das arquiteturas mais habituais dos servidores de computação atuais, nomeadamente daqueles baseados em dispositivos CPU *multicore* e *many-core*, explorando técnicas de programação vetorial e paralela; os dispositivos GPU podem ainda ser usados como aceleradores de algumas funções. Vários resultados de validação obtidos num servidor de computação são apresentados, em conjunto com algumas melhorias obtidas durante as fases de implementação e teste.

O *TomSeg* é uma ferramenta de código aberto para *Unix* e *Windows* que pode ser estendida facilmente com novos algoritmos para manusear de forma eficiente qualquer tipo de tomogramas.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

2D	Two-Dimensional.
3D	Three-Dimensional.
4D	Four-Dimensional.
ALU	Arithmetic Logic Unit.
API	Application Programming Interface.
CLI	Command-Line Interface.
CPU	Central Processing Unit.
CUDA	Compute Unified Device Architecture.
EM	Electron Microscopy.
ET	Electron Tomography.
FIB	Focused Ion Beam.
GPU	Graphic Processing Unit.
GUI	Graphical User Interface.
HPC	High Performance Computing.
INL	International Iberian Nanotechnology Laboratory.
JSON	JavaScript Object Notation.

MIC Many Integrated Core.
MIMD Multiple Instructions, Multiple Data.
MPI Message Passing Interface.
MTJ Magnetic Tunnel Junction.

NUMA Non-Uniform Memory Access.

RAM Random-Access Memory.
RAW Read After Write.
ROI Region of Interest.

SEM Scanning Electron Microscopy.
SIMD Single Instruction, Multiple Data.
SIMT Single Instruction, Multiple Threads.
SMX Streaming Multiprocessor.

TEM Transmission Electron Microscopy.

UI User Interface.
UMA Uniform Memory Access.

VOI Volume of Interest.

1

INTRODUCTION

Tomograms are widely used to represent *Three-Dimensional (3D)* data, and are essentially a stacked series of *Two-Dimensional (2D)* images. Each image represents a computationally generated slice of the 3D structure, whose elements are designated by *voxels*.

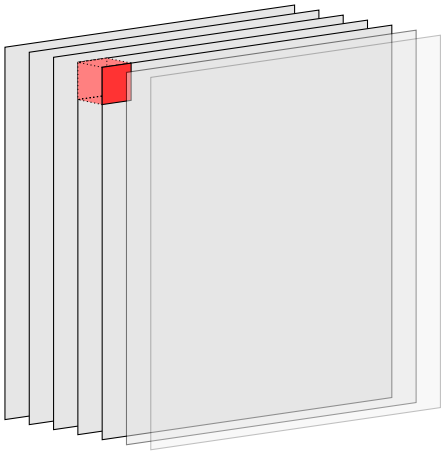


Figure 1.1.: *Voxel* of a 3D volume

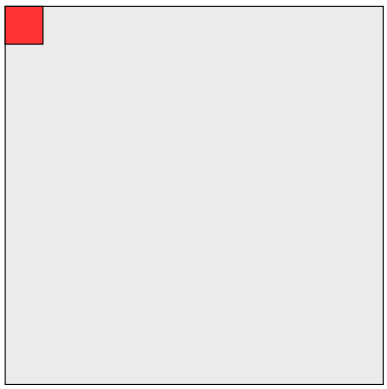


Figure 1.2.: *Pixel* of a 2D image

The method to obtain tomograms is called *tomography*, which is fundamental in areas such as materials science, radiology, archaeology, biology, plasma physics, and quantum information.

In the context of this dissertation the tomograms are specifically obtained through *Electron Microscopy (EM)* techniques, which are capable of imaging at a significantly higher resolution than light microscopes. Taking advantage of short wavelengths, they use beams of accelerated electrons as a source of illumination to reveal the structure of objects which are imperceptible using photons. Due to their versatility, electron microscopes are used to analyse organic and inorganic samples. Therefore, tomograms have been successfully

used to support studies related to nanosized materials, biology, and medicine. As a result, not only did these tomograms aided on the development of new microelectronics and renewable energy applications, but also revealed the structure of protein complexes.

Electron tomograms, even from extremely thin samples, can contain hundreds of stacked images replete with essential information. To reduce that complexity, as well as make quantitative analysis tractable, the tomograms of electron densities must be segmented into interpretable volume maps, where each new *voxel* directly identifies which feature of interest it represents.

Electron tomogram sizes are normally on the gigabytes' magnitude, so processing them is not a straightforward task. To develop useful algorithmic solutions to these problems, it is crucial to be aware of how accessing the data impacts on performance and to know how to hide this latency, by knowing the functioning of the accessing mechanisms.

What further complicates the segmentation process is the reduced signal-to-noise ratio of the images. On one hand, this is caused by the high complexity of the samples, on the other it comes from the potential of electrons to damage the sample, i.e., it is necessary that electron doses are high enough to obtain measurable contrast, but low enough to minimise sample damage.

1.1 CHALLENGES AND MOTIVATIONS

With the accelerated advances in equipment automation and in computational power, tomogram acquisition and reconstruction are no more the main difficulties to scientists. In contrast, the progress made on techniques to segment these tomograms was not sufficient, and the phases of feature extraction and qualitative analysis are now the current bottleneck of most electron tomography experiments. This absence of automated tools to extract features from tomograms is largely due to the difficulty of developing a generic and yet accurate method to process this large amount of data with such low signal-to-noise ratio.

Unlike 2D image processing, a field vastly explored and rich in software applications, processing 3D tomograms lacks efficient and widely accepted algorithmic foundations to give answer to the evolution of experimental science outcomes. Due to the increased complexity involved, specially when compared to 2D techniques, even the most basic operation like

handling or displaying 3D data in a user-friendly manner becomes a challenging task. This alone makes it difficult to design and validate an algorithmic solution that would segment and extract useful information from the tomograms, but doing so using efficient processes is still more challenging.

In addition to that, in situations when taking full advantage of the new programming paradigms and the different computing systems architectures is fundamental, it is no longer the algorithmic solution the exclusive purpose of analysis. The specific concepts of each paradigm and architecture must be present during the design of the solutions so that they can benefit and deal with the different particularities.

1.2 GOALS AND CONTRIBUTIONS

The aim of this work is to explore new automatic approaches of how to handle tomogram data, requiring minimal user interaction. With these results, several institutions with electron microscopes, namely the *International Iberian Nanotechnology Laboratory (INL)*, hope to increase their overall efficiency and reduce the bias introduced by their researchers when extracting and analysing the information from the tomograms.

As tomogram data are complex and normally large, high performance techniques must be addressed to produce results in useful time. Therefore, in addition to build a solution that correctly manipulates tomograms, one of the main objectives is to build it considering the characteristics and limitations of the computational platforms where it will be executed. This includes concepts ranging from how the data access latency is hidden in the most popular computer architectures, to how *Graphic Processing Units (GPUs)* and *Many Integrated Core (MIC)* devices can speedup the process by executing parts of the work.

The final result is *TomSeg*: a functional and efficient open-source software package (Chapter 4) capable of loading, displaying, aligning, cropping, segmenting, and exporting Slice and View tomograms. With its modular design, *TomSeg* can also represent a new software environment where different solutions can be easily developed and tested.[1]

1 *TomSeg* is available at `https://github.com/prsousa/TomSeg` under the GNU General Public License v3.

1.3 OVERVIEW

This dissertation is structured in five chapters after this introduction, summarised below:

Chapter 2, *Electron Tomography*, introduces the key concepts of electron microscopy and shows how it is related to tomography, presenting the common techniques to collect tomographic data. The usual workflow of electron tomography procedures is shown and discussed, focusing on the alignment and feature extraction steps.

Chapter 3, *Challenges for Efficient Computing*, addresses the analysis of the aspects of the computing platforms used. It is centred on the fundamental requisites for tomography, namely the memory hierarchy system present in common computer devices and the different parallel paradigms available on compute servers. The foundations for the conceptualisation of the solution are laid throughout this chapter with each concept.

Chapter 4, *The TomSeg Tool*, presents the solution built to the problems verified and analysed in the preceding chapters. This solution consists on a tool with several functionalities, which are deeply detailed across the chapter using several example images. A special focus is given to the new proposed process to segment the 3D data.

Chapter 5, *Validation and Performance Analysis*, tests the functionalities of the tool, discussing the produced results and evaluating thoroughly the performance metrics obtained with each. To do this, three representative tomograms are selected. These analyses are performed both in a personal laptop and in a compute server, demonstrating the adaptability of the solution.

Chapter 6 presents the final considerations about the process of building the tool, its strong points, and its not so strong characteristics. The chapter closes with some suggestions for future improvements to this work.

2

ELECTRON TOMOGRAPHY

2.1 ELECTRON MICROSCOPY

Electron microscopy techniques use accelerated electron beams as an illumination source. As electron wavelengths can be up to 100 000 times shorter than that of visible photons, electron microscopes have a higher resolution power than a light microscope, allowing them to reveal much smaller objects in finer detail.

Amongst electron microscopes, several types are available. Depending on the specifics of the experiment, one is more suitable than the others. In the context of this work, the datasets analysed were obtained using microscopes of *Transmission Electron Microscopy (TEM)* and, especially, of *Scanning Electron Microscopy (SEM)*:

TEM a broad beam of electrons is sent towards a thin specimen, forming a magnified image by detecting the electrons that pass through;

SEM a narrow beam of electrons is moved across the specimen, building an image pixel by pixel by detecting signals resulting from interaction of the electrons with the sample, as it moves.

Electron microscopes are becoming an essential part of the equipment in many laboratories and institutes. Researchers use them to examine different materials, ranging from inorganic nanocrystals to frozen slices of cells. The INL, specifically, has recently invested millions of euros to equip its laboratories with the latest technology in this field.

At the same time, these microscopes, being very versatile, are becoming popular in many contexts other than scientific research: currently, the major microelectronic production lines are equipped at least with one electron microscope; within forensics laboratories, they help

analysing samples such as gunshot residues; in the area of fault diagnosis and quality control, their use can range from evaluating stress tests in engine parts to verifying the porosity of conserved food.

2.2 TOMOGRAPHY

All forms of tomography are a means to reconstruct the 3D structure of an object from a series of 2D images. Even with theoretical results for *Electron Tomography (ET)*, it was not until the early 1990s that the computing power needed for volume reconstructions became available. Only then did practical applications of the technique begin to spread (Frank, 2008, p. 119). Two main EM techniques are presently used to collect projections of samples: the Tilt series (only for TEM images), and the Slice and View serial sectioning:

TILT SERIES is a non-destructive technique that tilts the sample and takes multiple TEM projections; it is commonly used in structural biology studies. Its basic functioning is shown in Figure 2.1: the sample is tilted around a single axis with small incremental angle steps and different TEM projection images are taken (McIntosh et al., 2005);

SLICE & VIEW is a destructive technique that slices the sample and takes multiple images; it is widely employed to detect and analyse defects in microelectronics. The technique uses *Focused Ion Beam (FIB)* serial sectioning as shown in Figure 2.2: an ion beam mills the sample (FIB) and the exposed face is then imaged with secondary electrons, usually using SEM techniques (Denk and Horstmann, 2004).

After collecting the data, the obtained projection images must be aligned and the 3D volume reconstructed. Only with the volume correctly reconstructed is it possible to extract the different features and to perform a qualitative analysis. The full workflow proposed by Tsai et al. (2014) includes a pipeline of four phases:

RAW DATA COLLECTION a set of TEM/SEM techniques used to image the sample;

DATA ALIGNMENT AND RECONSTRUCTION two different approaches can be used to align the slices based on either TEM Tilt series or Slice and View serial sectioning (more details in Section 2.3);

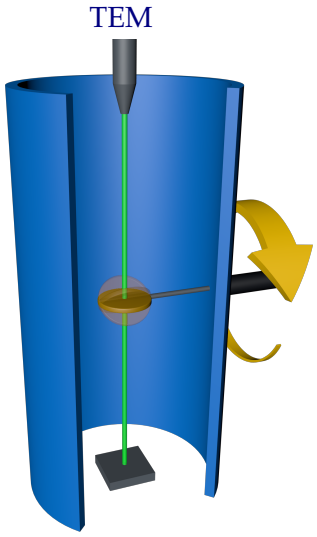


Figure 2.1.: TEM Tilt series

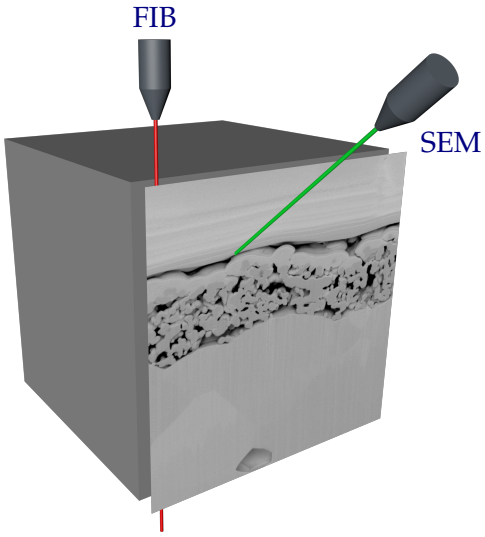


Figure 2.2.: Slice and View serial sectioning

NOISE REDUCTION AND CONTRAST ENHANCEMENT a preliminary phase to aid the key step to extract features from the tomogram: the 3D segmentation;

FEATURE EXTRACTION AND QUANTITATIVE ANALYSIS several operations can be identified in feature extraction and quantitative analysis, but all require a 3D segmentation (detailed in Section 2.4); the quantitative analysis includes tasks such as determining the porosity of a volume and the fraction of one material over another.

The challenges for the development of the current version of the new tool, the Slice and View *TomSeg*, basically follows the latter 3 phases of this workflow.

2.3 DATA ALIGNMENT AND 3D RECONSTRUCTION

The obtained image records have different characteristics depending on the tomographic method used. Hence, different methods must be applied to reconstruct the 3D structure.

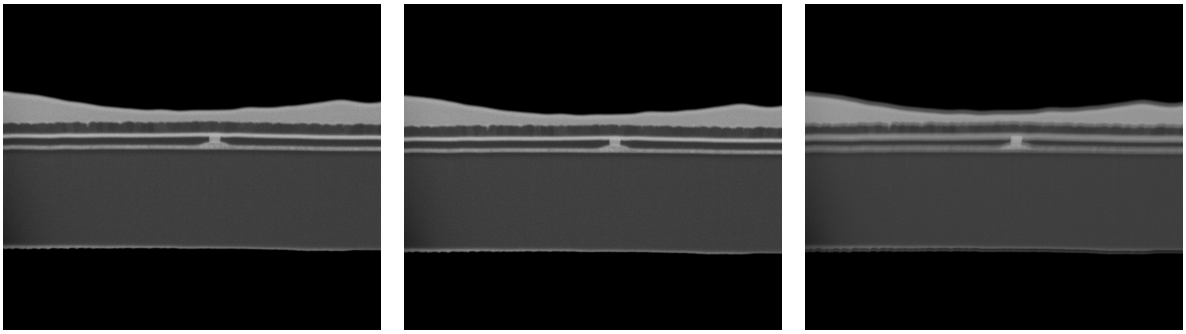
On TEM-Tilt series (Figure 2.1), the microscope records projection images containing information from all regions of the sample through which the beam was transmitted. Each image contains information from all heights of the sample collapsed onto a single plane. To describe the 3D structure, the sample is tilted and imaged using different angles. Since tilted images are projection-based, the volume cannot be reconstructed by simply combin-

ing them. Over the years several algorithms have been developed to overcome this difficulty. *Back-Projection* is one of the most accepted (Lewitt, 1983). Current Slice and View *TomSeg* is already structured to receive, test and validate these additional features in a future release.

When the acquisition method used is Slice and View (Figure 2.2) an almost direct map between the physical image and the reconstructed 3D slice can be made. This holds true because the images were obtained by milling and imaging the sample multiple times on the 3rd dimension. However, the images must still be aligned to correct stage movement and instability of the beam. In addition, as a consequence of using different milling and imaging resolutions, it is common to have higher resolution in either *x-y* or *z* directions (Bradshaw and Stahl, 2015, p. 47).

2.3.1 *Alignment by Cross-Correlation*

Cross-correlation alignment methods are based on the match between structural features of two images. As long as some of these structures are present in both images, the potential for a highly accurate alignment is evident. As an example, the Figure 2.3 shows two consecutive slices obtained using a Slice and View technique, in which a misalignment is visible, especially, along the vertical axis.



(a) Slice A (b) Slice B (c) Slices A & B overlapped

Figure 2.3.: Two misaligned slices

The highly accurate alignment of the collected images is one of the most fundamental prerequisites of tomography (Frank, 1992, p.205). Several alignment techniques are available to be used in electron tomography, but the use of gold particles as reference markers and

the calculation of the 2D cross-correlation function — both separately and in conjunction — are the most commonly used.

In the absence of artificial reference markers, two solutions are available: apply the cross-correlation between the whole images; or select a portion of the first image and use it as a reference marker.

Since the tomogram of Figure 2.3 presents a very well defined feature of interest along the two axes (Figure 2.4), the latter method is preferable. The normalised result of calculating the cross-correlation between the reference area and the second slice is shown in Figure 2.5. Regions with higher factor of signal matching are represented with higher intensities in the cross-correlation image.

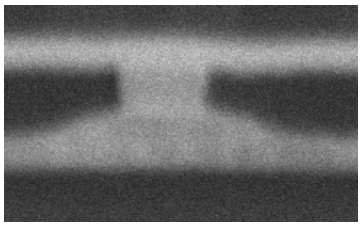


Figure 2.4.: Structural feature of reference

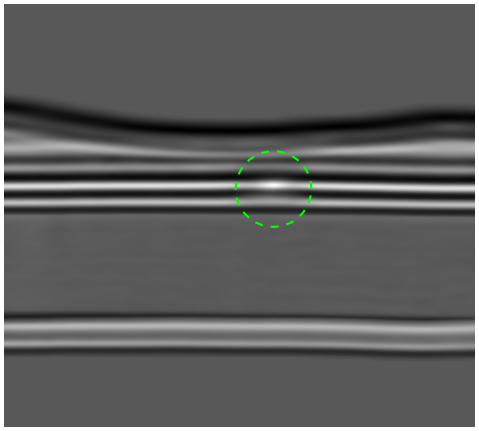


Figure 2.5.: Cross-correlation result

The point in which the correlation is at its maximum corresponds to the position of the reference feature in the second slice. Thus, by comparing its coordinates to the originals, it is trivial to compute the misalignment value between the slices.

2.4 3D IMAGE SEGMENTATION

After aligning the projections and reconstructing the 3D volume, there is a need to extract useful information from the tomograms. One approach to simplify the data is to use image segmentation techniques.

Image segmentation is defined as the mathematical process of partitioning a digital image into multiple segments of non-overlapping, adjacent regions (Pal and Pal, 1993). These

segments have some characteristics in common (e.g. intensity, colour, texture, proximity), which, when accurately defined, can provide a meaningful and simplified representation of the information within the original image. The Figure 2.6 shows an example of a EM image and, to its right, a corresponding segmentation result.

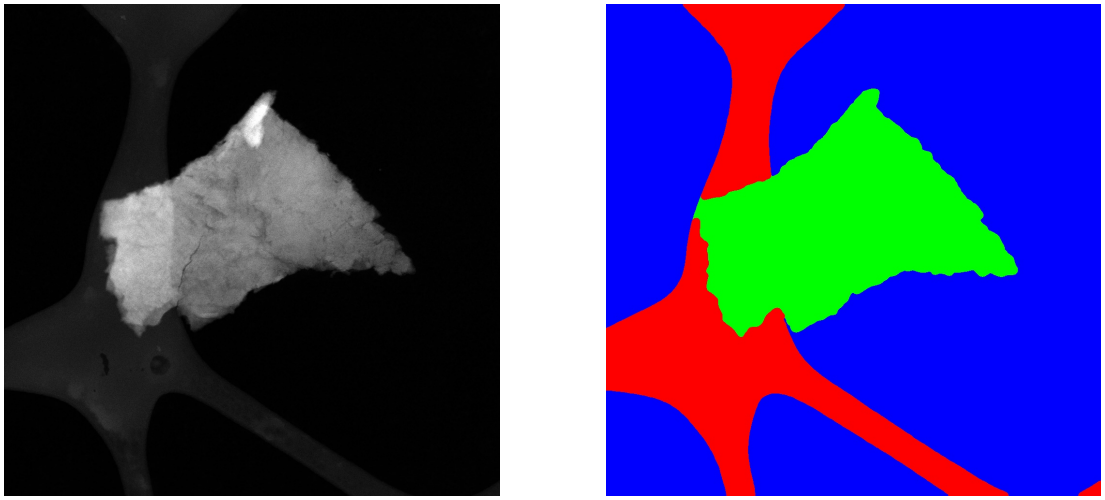


Figure 2.6.: Example of image segmentation (obtained with *TomSeg*)

Currently, there are several segmentation techniques capable of automatically perform that task. However there is no single method which can be considered adequate for all images. As expressed in the work of Tsai et al. (2014), with available methods, specific expertise is often needed to interpret complex tomograms and to focus on the essential components (or even on the important regions) of the 3D volume. What further complicates the analysis is the fact that visualisation of 3D volumes is remarkably nontrivial.

Recent researches classify current techniques of tomogram segmentation in four distinct groups, suggesting that, depending on the expected results, work effort, and tomogram characteristics, one should be selected (Tsai et al., 2014).

2.4.1 *Manual Methods*

Segmentation is designated manual when the recognition of interesting objects, and its respective delineation on an image, is carried out by a human operator (Garduño et al., 2008). Despite of the efforts on developing automatic solutions, manual segmentation continues to be the preferred method in ET (Wei et al., 2012). It is sometimes considered as the

only practical approach for segmenting heavily noise contaminated tomograms (Ruhaiyem, 2014). Unfortunately, fatigue and user bias are reflected on the results produced.

Manual Abstracted Model Generation

Manual abstracted model generation is the first class of segmentation techniques introduced by Tsai et al. (2014), being also the simplest one. It is particularly effective to segment linear objects. After providing some seed points, an automatic tool is then used to connect them. As shown in Figure 2.7, the produced results can enhance feature's length and orientation measurements. They present an acceptable abstracted model for both qualitative and quantitative analysis.

This technique is often resorted to in experiments in which reducing resources spent on the analysis is more important than being absolutely loyal to the objects on the tomogram. It is important to note that, as long as the human eye can perceive the objects of analysis, data contrasts, crispness, and crowdedness do not majorly influence this methods' success.

Figure 2.7 shows a manual abstracted model of a mitochondrion. As expected, rather than a exact volume model, it represents a skeleton version of the 3D densities.

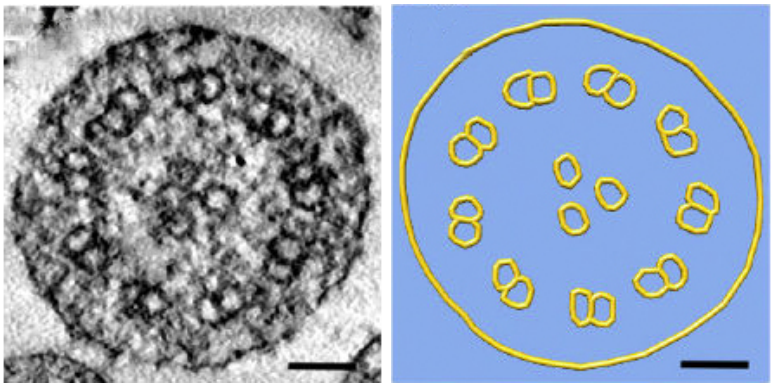


Figure 2.7.: 3D Manual Abstracted Model (Tsai et al., 2014)

Manual Tracing-Segmentation & Surface Rendering

The second manual method class described by Tsai et al. (2014) consists on paint brushing different slices, tracing the features in the 3D space. With software support it is possible to create an interpolation between intermittently segmented slices, leading to smooth changes on the rendered surface.

Manual tracing succeeds on almost all datasets. Nonetheless, it is the most time-consuming method. Frequently, it is the only technique to extract useful information from complex image sets enclosing a large diversity of features, such as with the thin cell membranes. Manual tracing can be applied very efficiently if the data are crisp and have satisfactory contrast. Moreover, as long as the user is familiar with the objects under analysis, this method can also be applied in more demanding datasets.

Figure 2.8 shows the result of manual tracing-segmentation of the same mitochondrion dataset. Comparatively to its abstracted model, a partial 3D model was now generated, allowing other sort of spatial analyses.

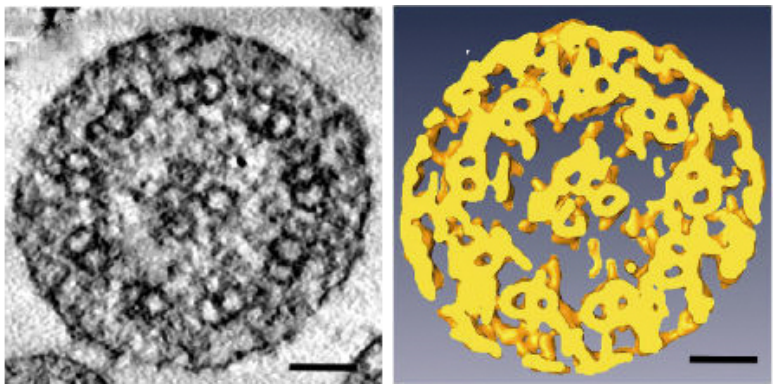


Figure 2.8.: 3D Manual Traced segmentation (Tsai et al., 2014)

Semi-Automated Segmentation & Surface Rendering

Unlike the fully manual methods, semi-automated segmentation approaches are typically less time-consuming. This turns viable to consider larger stacks of images. However, if the methods are not thoroughly selected and configured, a lot more time may be necessary to refine and curate the segmented volumes.

Automated density-based segmentation works best on datasets with a large number of similar features of interest, all requiring segmentation. More complex datasets may also be targeted, but only as an initial step, requiring a user intervention in order to choose the interesting segmented volumes.

Results shown on Figure 2.9 are identical to the ones obtained using manual tracing segmentation. The main difference from the previous method lies on the model generation process and its associated work effort.

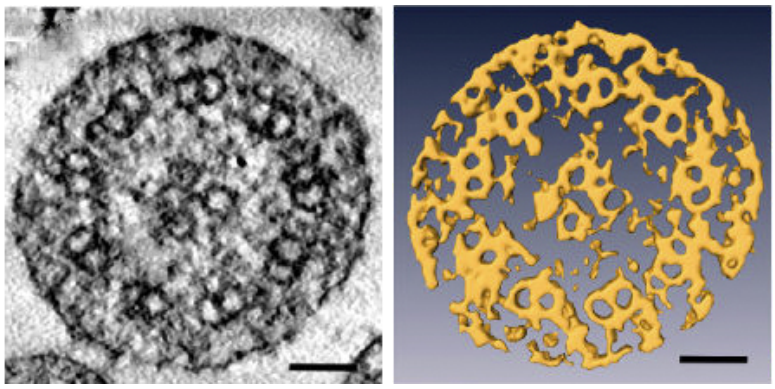


Figure 2.9.: Semi-automated 3D segmentation result (Tsai et al., 2014)

2.4.2 *Automated Algorithms*

With the power of algorithmic customisation, automated segmentation methods eliminate the user bias and dramatically increase the process efficiency. But, with current solutions, they are often appropriate for only a limited number of feature characteristics and datasets.

Prior to automated segmentation, an effective procedure is filtering the images to reduce its noise. The 3D median filter and the non-linear anisotropic diffusion filter were proven to be applicable for reducing noise from tomograms, improving significantly the signal-to-noise ratios (Narasimha et al., 2008).

Thresholding

The simplest property that voxels of the same class can share is intensity. A way to segment such regions is through a technique called *threshold*. This technique creates a binary image by turning all voxels above a threshold value to one (1), and all the remaining ones to zero (0), as defined on Equation 1.

$$g(x,y,z) = \begin{cases} 1, & \text{if } f(x,y,z) > T \\ 0, & \text{if } f(x,y,z) \leq T \end{cases} \tag{1}$$

g: resulting binary image

f: original image

T: threshold level

This technique is simple and intuitive, but has the problem of just considering the voxels' intensity, ignoring the relationships between them. It is not aware of class continuity.

Nevertheless, several ET experts continue to classify thresholding as the most capable method of automatic image segmentation (Joos et al., 2011). Unfortunately, the majority of its results are only acceptable with images presenting a bimodal distribution of its grey-scale values. In those circumstances, the unique difficulty is to find the best threshold value.

Among the automatic methods to determine the threshold value, the solution proposed by Otsu (1975) is still one of the most common in ET segmentation. It aims to maximise the separability of the resultant classes in grey-levels, trying to divide the grey-scale into the two most distant clusters possible.

Figure 2.10 shows a 2D image obtained using EM techniques and its respective grey-scale histogram. To this example it is considered that the image just presents two main classes: the thin material, and the background.

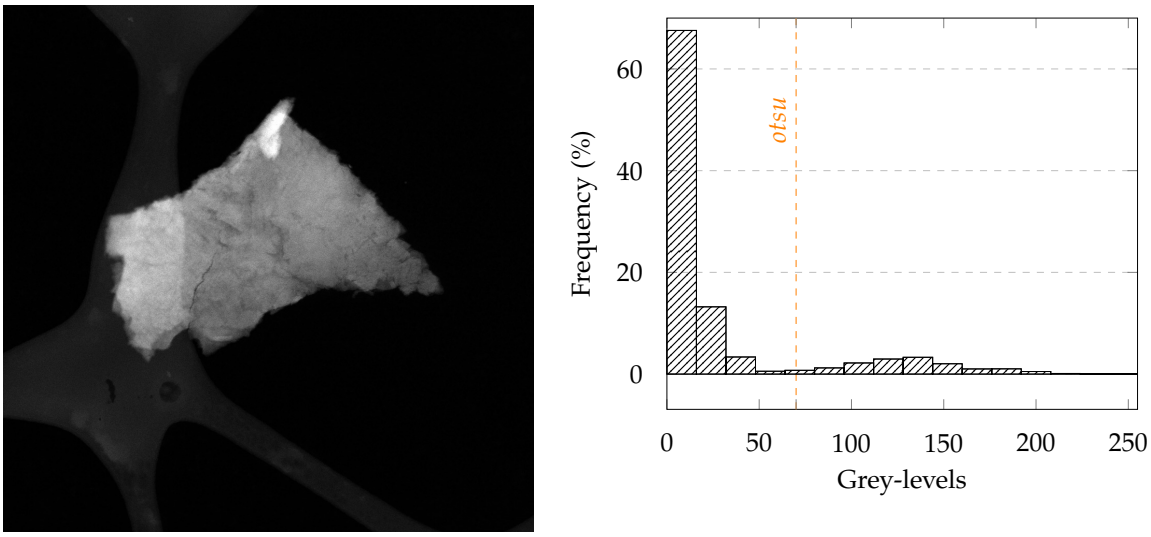


Figure 2.10.: A 2D EM image and its grey-level histogram (w/Otsu value)

Applying the algorithm proposed by Otsu, the threshold value was set to 70. The consequent segmentation results are shown in Figure 2.11.

Considering only these two classes of pixels, the resulting segmentation proved to be satisfactory. However, by focusing on the pixels at boundary areas between the two classes, one can notice some imperfections, which resulted on the creation of a large number of

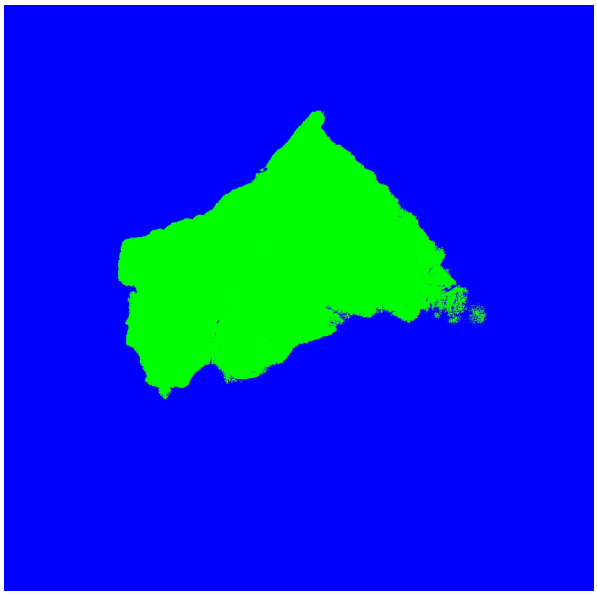


Figure 2.11.: Segmentation of a 2D EM image using its Otsu value

distant, disconnected, regions. This problem becomes larger when the image histograms are more uniform and when more classes are considered within the images.

Seeded Region Growing

Methods based on *seeded region growing* segment the images with respect to a set of initial points (or areas), known as *seeds*. In their definition, Adams and Bischof (1994), propose an algorithm where regions grow in an iterative fashion from the initial seeds: at each iteration all pixels at the border of the growing region are examined and the most similar to the region is appended, repeating this operation until all of the image pixels have been assimilated.

When compared to threshold and boundary-based techniques, region-based methods enable the definition of multiple assimilation criteria and, even more importantly, can benefit from the knowledge about voxels' spatial location. This latter characteristic is absolutely relevant, as it allows to consider physical continuity between the voxels of the same class. However, these methods depend on the selection of *seed* points. Depending also on the assimilation criteria, a convergence to the final result may never occur. The segmentation procedure of the *TomSeg* tool takes advantage of methods based on region growing to expand the regions as much as possible. This method is a fundamental part of the process,

because it allows to consider the continuity of the voxels that belong to the same class. Its assimilation criteria are based on custom histogram analysis, which are explained in detail in Chapter 4.

As an example, the previous EM image is shown again in Figure 2.12, but now considering three different classes of pixels. It is important to note that in these methods the user has to identify each class with a *seed* area: the thin material (green), the support film (red), and the background (blue).

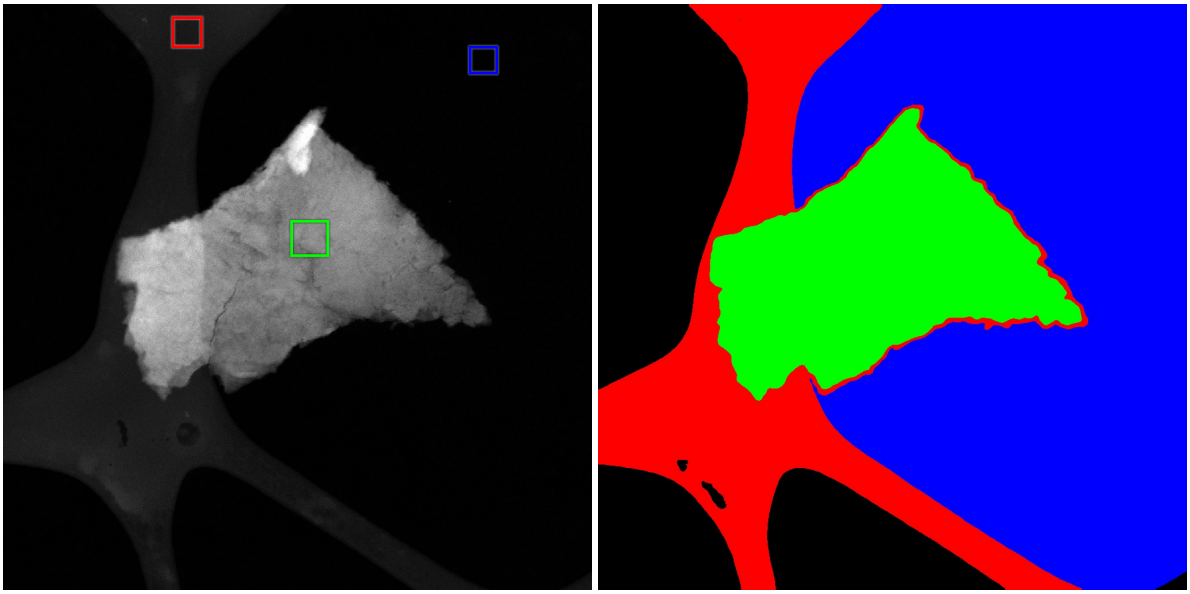


Figure 2.12.: Segmentation of a 2D EM image using a region growing method

Comparing to the previous thresholding results, the segmentation results obtained using this region growing based procedure (with a custom assimilation criteria) are much more satisfactory. In addition to being able to consider an arbitrary number of classes, the notion of continuity and neighbourhood between pixels is considered in the process of assimilation. It is important to note that, given the expansionary nature of the algorithm, regions of the image that are not reachable through the *seed* areas will not be visited, and, therefore, will not be segmented.

A more complex algorithm must be designed to fully segment the images. Once the first flood ends, one possible approach is to automatically search for new *seeds* in the unvisited regions and expand from there using a similar (or even the same) method. The *TomSeg*'s segmentation algorithm is based on this concept (details in Chapter 4).

Watershed

The watershed transform was developed to segment biological specimen accurately, where images are considered as topographic surfaces and the gradient magnitude is seen as the elevation. It has technically proven its ability in detecting object regions based on region's grey intensity (Volkmann, 2002). The original design had some limitations, such as over-segmentation (Beucher and Meyer, 1992, p. 446), which motivated the development of alternative versions, relying on initial seed points to start flooding the image.

Even not used directly, the overall functioning of the algorithm and its main components were an important inspiration to design the main segmentation algorithm of *TomSeg*.

2.5 CURRENT SOFTWARE TOOLS

Currently there are some software packages to handle tomograms. Some of them are open-source, but the most used and complete ones are proprietary. In the context of this project four of them were analysed in terms of the main features, the capability of third-party integration, their native awareness of computing performance and, above all, their capabilities to automatically segment tomograms.

AMIRA is an extensible software system for scientific visualisation, data analysis, and presentation of 3D and *Four-Dimensional (4D)* data (Stalling et al., 2005). It takes advantage of multithreading computing when rendering the images and provides basic image segmentation tools. It is available to buy an extra software extension, XImagePAQ, for advanced image processing and quantification, providing features to automatically segment biological volume images.

In the context of this work it was not possible to integrate or expand any of its functionalities because it is a proprietary software.

IMOD is a set of computer programs that allow the user to display, process, and produce models from images (Kremer et al., 1996). These tools were primarily developed at the University of Colorado and are currently open source.

Although it is quite complete for the tomographic alignment and reconstruction phases, it only provides some basic filtering and edge detection algorithms to the segmentation and feature extraction phase.

TomSeg was not built as an extension to IMOD, since its huge amount of tools and its well-defined paradigm turned out to be inadequate to test and validate a different segmentation approach. However, its software architecture and the versatility of its command line utilities served as inspiration to build *TomSeg* as it is today.

CHIMERA is an extensible program for interactive visualisation and analysis of molecular structures and their related data (Pettersen et al., 2004). It is currently developed and maintained at the University of California under an open source licence.

Featuring only a few basic tools for manual segmentation, the Chimera's most interesting feature is the extremely easy and intuitive way in which allows the user to view, handle, animate, and export 3D images.

In the context of this work, this package is very useful to visualise the segmentation results produced with *TomSeg* in the 3D space (Figure 2.13). By importing the resulting `mrc` files (details in Appendix A) describing the segmented volume, and taking advantage of its GPU accelerated rendering engine, Chimera provides a smooth and precise experience, which is ideal to perform the posterior qualitative analysis.

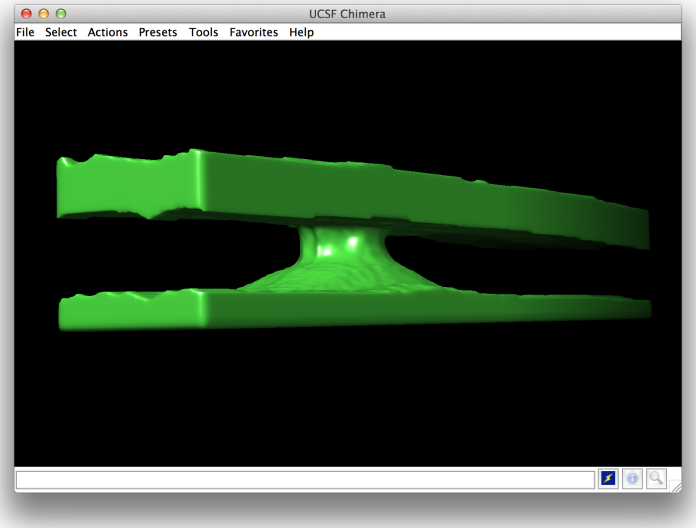


Figure 2.13.: *TomSeg* result visualised in 3D space using *Chimera*

IMAGEJ is a Java application designed to handle scientific multidimensional images, distributed under an open source program (Collins, 2007).

It supports standard image processing functions such as contrast manipulation, sharpening, smoothing, edge detection, and median filtering.

An interesting characteristic is its multithreading computing support, taking advantage of this common feature of modern computer architectures to speedup the overall execution.

3

CHALLENGES FOR EFFICIENT COMPUTING

The current trend in computing platforms is centred on a key set of features.

With the crescent gap between processing and memory access speeds it is critical to find solutions to reduce, or hide, the memory latency: some manufactures opted for a memory hierarchy with very fast cache levels; others opted to support, with hardware, a very large number of threads.

To respond to the growing needs of digital signal processing and graphics processing, the technology companies, inspired on the the *Single Instruction, Multiple Data (SIMD)* super-computers, have been including vector based approaches in their architectures to explore data parallelism.

At the same time, with high clock frequencies and facing heat dissipation problems, some manufactures opted to simplify and configure the processing units in multiple-core devices sharing a common memory, or in more than one multi-core device sharing separate memories (*Non-Uniform Memory Access (NUMA)*). More recently, devices with a very large number of *cores* (MIC) are also becoming popular.

3.1 DATA ACCESS LATENCY

The memory system is the repository for all the information used and produced by the processing units. The constant increase on microprocessors performance places a significant demand on the memory system. A perfect memory system would be one that could supply immediately any datum that the processing units requested. However, as memory capacity,

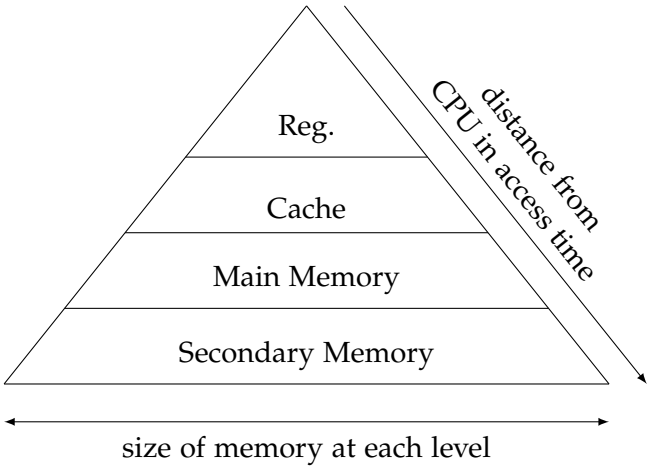


Figure 3.1.: Structure of a common memory hierarchy

speed, and cost are in direct opposition, this ideal memory is not practically implementable (Mahapatra and Venkatrao, 1999).

3.1.1 *Memory Hierarchy*

Generally, programs exhibit both temporal and spatial locality. Temporal locality is the tendency to reuse recently accessed data items; spatial locality is the tendency to access data items that are close to other recently accessed items. These two characteristics, together with the increasing gap between the processor and memory speeds, lead several companies to develop a complex memory hierarchy to hide the memory latency (Patterson and Hennessy, 2013, p. 450).

A system that relies on a memory hierarchy to hide the memory latency takes advantage of smaller, but faster, memory technologies. These *cache* memories are also physically closer to the processor, and may even share the silicon area of the processing electronics. However, being limited in capacity, they only store replicas of a tiny sub-set of a larger lower-level memory, as shown in Figure 3.1. Thus, accesses that *hit* in the highest level of the hierarchy can be resolved quickly, while accesses that *miss* pass to the lower-levels of the hierarchy, which are larger, but slower.

Once it is present in most of the current computer systems, it is critical to be aware and to take the proper advantage of this mechanism. For that very reason, spatial and temporal

locality of the data accesses is a desirable characteristic that must be ensured whenever possible.

Tomogram Slices

The memory hierarchy mechanism was a key issue while designing each of the *TomSeg*'s segmentation algorithms. Since tomograms can be seen as stacks of 2D image, processing tomograms can very easily compromise the performance of the application if it does not take advantage of data locality. Thus, and being also aware of the multithreaded computing features presented in Section 3.3, a decision was made to orient the core and hard segmentation processes to the 2D slices, trying to avoid the inspection on the 3rd dimension to correctly form the segmented volume (more details in Section 4.5).

3.1.2 *Multithreaded Parallelism and Fast Context Switching*

A different approach to hide data access latency is to execute many instances of the same or different programs at the same time. Multithreading is widely used in commodity processors, especially in GPUs, since it allows a better use of their replicated and deeply pipelined resources (Ryoo et al., 2008).

With the very fast context switching support of GPUs devices, when a memory fetch is issued while processing a subset of the data elements, the scheduler puts the corresponding *warp* aside in favour of another that is not waiting on a memory reference. However, doing this switch so quickly requires an enormous amount of registers so that the context of all *warps* are always loaded and ready to execute at any moment.

Figure 3.2 shows a scheme of the *Nvidia Kepler GK110* architecture, which is organised in 15 *Streaming Multiprocessors (SMXs)*. In this architecture each of them is composed by a 256 KiB register file, meaning that the whole chip contains 983 040 registers of 32 bits.

In the same figure it is also visible the existence of *multilevel caches*. But, unlike in the *Central Processing Units (CPUs)*, their aim is not to hide the long latency to memory. Instead, GPU's memory system is oriented toward bandwidth.

Figure 3.2.: Architecture of Nvidia Kepler GK110 (Nvidia, 2012)

3.2 VECTOR COMPUTING

The basic data type in a traditional scalar processor is a `n`-bit *word*. The architecture often exposes a register file of *words* and the instruction set is composed of instructions that operate on individual *words*. Vector architectures support a *vector* data-type, where a *vector* is a collection of `N` *words*. There may also be a *vector* register file, which was a key innovation of the Cray architecture (Russell, 1978).

$A7$	$A6$	$A5$	$A4$	$A3$	$A2$	$A1$	$A0$
			+				
$B7$	$B6$	$B5$	$B4$	$B3$	$B2$	$B1$	$B0$
			=				
$A7+B7$	$A6+B6$	$A5+B5$	$A4+B4$	$A3+B3$	$A2+B2$	$A1+B1$	$A0+B0$

(a) SIMD mode (`N` elements at a time)

A
+
B
=
$A+B$

(b) Scalar mode (1 element at a time)

Figure 3.3.: Comparison between SIMD and scalar operations

3.2.1 *As Extensions to Computer Architecture*

A variation of the SIMD machines operation can be found in almost every today's microprocessors. With the emergence of multimedia applications, manufacturing companies have begun to extend their instruction sets to increase the performance of their chips. These instructions allow the hardware to have multiple *Arithmetic Logic Units (ALUs)* operating simultaneously over different data elements (Patterson and Hennessy, 2013, p. 648).

However, the multimedia extensions added to the common computer architectures have some limitations when compared with the pure vector architectures. Multimedia extensions typically specify fewer operations when compared with vector computers. Unlike multimedia extensions, the number of elements in a vector operation is variably defined in a separate register. This means a different version of the vector architecture can be implemented with a different number of elements just by changing the value of that register. In contrast, in the multimedia extension architectures, a new large set of instructions needs to be added each time the *vector* length changes.

Finally, but very important in the context of *TomSeg*, the lack of support in the multimedia extensions to strided and indexed data accesses forces the programmer to be aware of how the data is accessed, constricting, once again, the final solutions.

3.2.2 *In Graphic Processing Unit Devices*

The Nvidia's GPU architectures handles massively parallel applications by grouping and scheduling the different concurrent threads in a clever manner. As schematised in Figure 3.2, the hardware is divided in several SMX units, which can be seen as *vector cores*. To maximise the performance, each *Compute Unified Device Architecture (CUDA)* core of a SMX must execute the very same instruction over different data elements at a time. Such approach is known as *Single Instruction, Multiple Threads (SIMT)*, which can be slightly compared to a vector architecture, considering a thread as a data unit. To process the data, each of the 15 SMXs of Nvidia's GK110 is equipped with 192 single-precision CUDA cores, 64 double-precision CUDA cores, 32 special units, and 32 load/store units.

It is important to note that, in contrast to vector architectures, which rely on compilers to recognise data-level parallelism at compile time to generate vector instructions, the most common GPU devices exploits the data-parallelism among threads in runtime.

3.3 MULTIPROCESSING UNITS

To address the growing need to solve harder problems and to speed the existing solutions, computing systems became more complex and more accelerated. However, the power limit and heat dissipation problems have forced a dramatic change in the design of microprocessors (Patterson and Hennessy, 2013, p. 41).

The strategy adopted by major hardware companies was to simplify their microprocessors and pack multiple processing units on a single chip. Nowadays, even the most basic computing system is composed by multiple processing units, designated by *cores*.

3.3.1 *Multiprocessing Paradigms*

Multiple Instructions, Multiple Data (MIMD) systems allow the simultaneous execution of different instructions on different pieces of data. These systems are organised in two major categories, according to how data is accessed and shared:

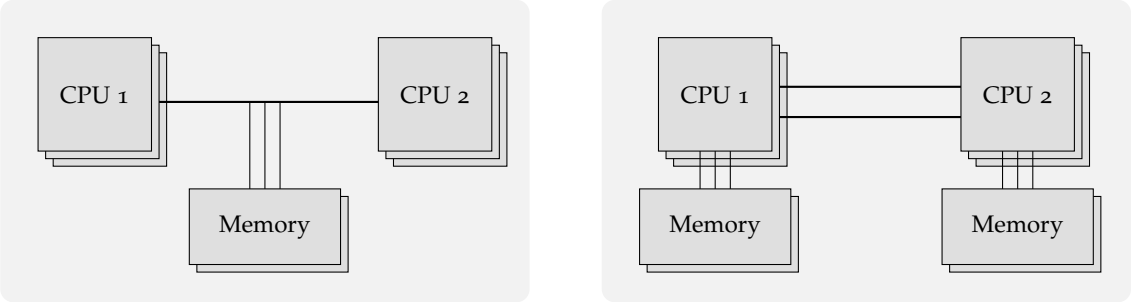
SHARED MEMORY the different threads share a common address space — the most preferred approach for parallel programming.

In shared memory systems, two further sub-categories can be identified: *Uniform Memory Access (UMA)*, and NUMA (Patterson and Hennessy, 2013, p. 638). With the former, the processing units experience the same time to access any memory address, while with the latter, the access time may vary according to the memory address. Typically, there is a distinction between accessing a local memory reference and accessing a reference stored in a *Random-Access Memory (RAM)* device not directly connected to the local CPU-chip.

UMA is still the typical memory organisation in personal computers: all threads have an uniform access time to any memory bank.

On the other hand, the NUMA model gets relevant on compute servers and in cluster environments. There, computing nodes with more than one CPU-chip, and more than one RAM device, are very common. The non-uniform access occurs when the CPU-chips involved include the memory controllers on-chip. As a result, different memory banks are connected to distinct CPUs. Cross memory bank accesses are still possible due to proprietary interconnects between the CPUs, but they are slower than accesses to nearest memory banks.

Figure 3.4 shows these two different models on a dual-CPU node.



(a) UMA node (b) NUMA node

Figure 3.4.: Scheme of two shared memory models

In the last years several libraries came up to aid the development of efficient parallel applications. *OpenMP* is one of the most important library to shared memory parallel programming in C, C++, and Fortran (Dagum and Menon, 1998). It consists of a set of compiler directives, library routines, and environment variables, defining a portable and scalable model with a simple and flexible interface for developing parallel applications on platforms from personal desktops to supercomputers.

DISTRIBUTED MEMORY each process operates on its own physical address space.

The private memory space is either due to the specific programming model, or to a physical separation of the processes. This is very common when working with acceleration devices, namely the GPUs, and in a multi-node environment (Patterson and Hennessy, 2013, p. 641). Figure 3.5 shows an example of an heterogeneous environment with both accelerators and multiple interconnected nodes.

The data sharing is performed via explicit message passing techniques, which adds an extraordinary overhead. This can seriously influence the performance of the solutions. However, this is often the only paradigm that can deal with large-scale problems.

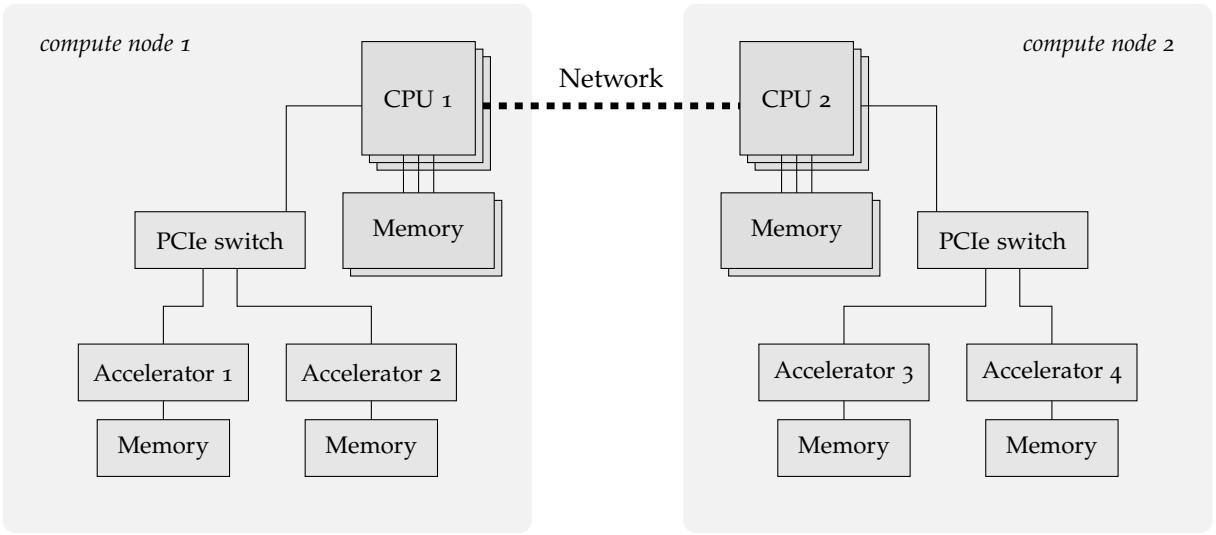


Figure 3.5.: Scheme of a distributed memory system

OpenMPI is an open source *Message Passing Interface (MPI)* library for developing distributed memory applications (Gropp et al., 1999). It exposes an *Application Programming Interface (API)* with a wide range of primitives for inter-process messaging passing and explicit synchronisation. The communication can be collective (*barrier*, *broadcast*, *gather*, *scatter*, *scan*, *reduce*, etc.), where a group of processes are involved, or point-to-point (*send*, *receive*), where only two processes participate. Additionally, several variants of these routines are available to support both blocking and non-blocking communication.

The inter-process communication is performed over network protocols, where the *Ethernet* is the most common. However, specific proprietary communication standards, namely *Infiniband* and *Myrinet*, can be used instead, to improve the performance of *High Performance Computing (HPC)* applications. The communication between processes running on the same compute node is often implemented using shared memory.

3.3.2 *Work Decomposition*

To take advantage of the multiple processors available on the current computing systems, the applications must explore the parallelism explicitly. To do so, it is important to decompose the program down into tasks, the smallest exploitable unit of concurrence. Each individual task is then assigned to a thread or process to be concurrently executed.

TomSeg, by handling tomograms as a series of 2D images, can easily decompose its segmentation process into several different tasks. Based on the concept of *seeded region growing*, presented in Section 2.4, the segmentation of sequential sub-volumes can be performed concurrently in different threads as long as each sub-volume knows its initial *seeds*.

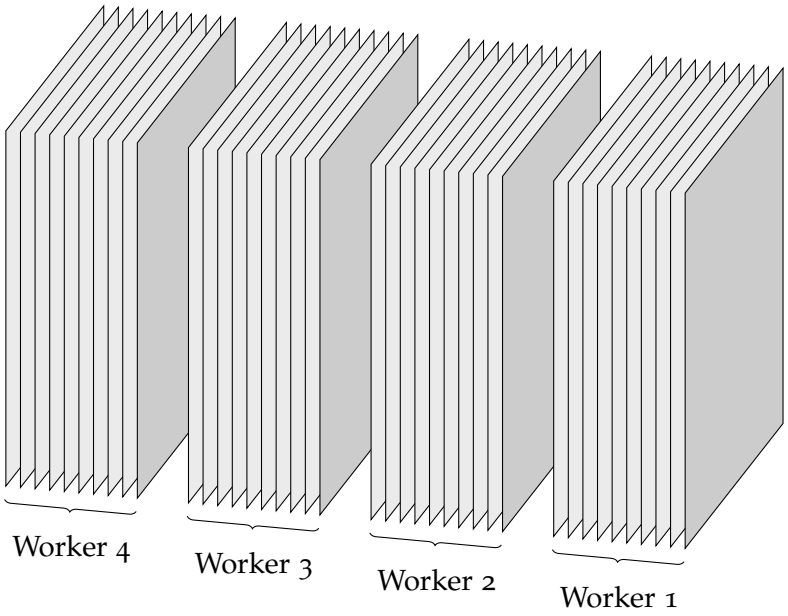


Figure 3.6.: Tomogram decomposed into four independent chunks

Figure 3.6 illustrates an example where a tomogram with 40 slices is decomposed and processed across 4 different independent workers. During the segmentation process, *TomSeg* splits the volume automatically at each *seeded slice*[1], creating the independent sub-volumes. The *seeds* can either be introduced manually by the user, or by a fully-automated procedure (detailed in Section 4.5) developed specifically for this purpose.

1 *Seeded slices* refers to the tomogram slices in which small regions of the image (known as *seeds*) indicate which class they represent.

3.4 TARGET PLATFORMS

Nowadays, researchers and scientists handle their tomographic results on their lab and own personal computers. Even being common computers, most of the concepts presented in the last sections are crucial to their performance. This work focused on the development of an application that could use the whole power of the systems, aiming to take the maximum advantage of three key aspects: the memory hierarchy, the multi-core architecture, and the vector extension instructions. Many of these common computing systems are also equipped with GPUs to accelerate graphic applications. Aware of this, when they are available, *TomSeg* accelerate the segmentation process, offloading some steps to the GPU.

On the other hand, another requisite to *TomSeg* is the capability to run in a server environment, where CPU-chips with dozens of *cores* and a large variety of accelerator devices are available. In this topic, some preliminary tests to *TomSeg* were successfully performed on Intel *Xeon Phi Knights Corner* coprocessor and also on the new *Xeon Phi Knights Landing* processor chip.

4

THE TOMSEG TOOL

The decision to develop a new tool to handle electron tomograms was taken after evaluating the available software packages and their limitations, targeting the main goals of this work: to build an efficient software tool to automatically segment Slice and View electron tomograms. The existing software tools address a very broad set of features, which, given their architecture design, makes it difficult to develop and test new ideas.

The development started as a basic C program to validate some ideas, but it quickly became a more robust package. Now, *TomSeg* is a cross-platform software package, available at GitHub (together with an installation guide) under the GNU General Public License v3.

TomSeg was developed in C++, using functions from the OpenCV library to load tomogram slices and the Qt framework to support the *Graphical User Interface (GUI)* components. It was designed to be a lightweight and modular software tool, where different algorithmic solutions to handle tomograms can be easily added and validated.

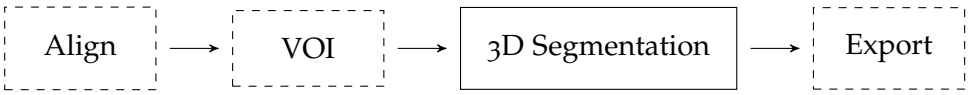


Figure 4.1.: The usual workflow on *TomSeg*

The current state of the tool under development, *TomSeg* v1.0, is based on the workflow of ET procedure proposed by Tsai et al. (2014), namely:

- Alignment of Slice and View serial sectioning tomogram slices, producing as output the reconstructed tomogram;[1]

1 Alignment and reconstruction of TEM Tilt series tomograms require a different algorithm, which can be added to *TomSeg* in a future version.

- Selection of the *Volume of Interest (VOI)*;
- 3D segmentation of Slice and View tomograms, including noise reduction features;
- Export the simplified 3D volume in MRC file format, allowing external visualisation in available tools.

4.1 Software Architecture

The first big decision was to develop the tool using an high performance object oriented language, C++. With a lot of entities and logic concepts, principles as the *separation of concerns* are fundamental to develop an *easy-to-debug* and modular platform. The *interface* and *abstract class* concepts are also a fundamental key to keep the code clean and easy to maintain/extend. At the same time that C++ offers high-level abstractions, since its applications are compiled into native machine code, it also allows the programmer to benefit from low level operations.

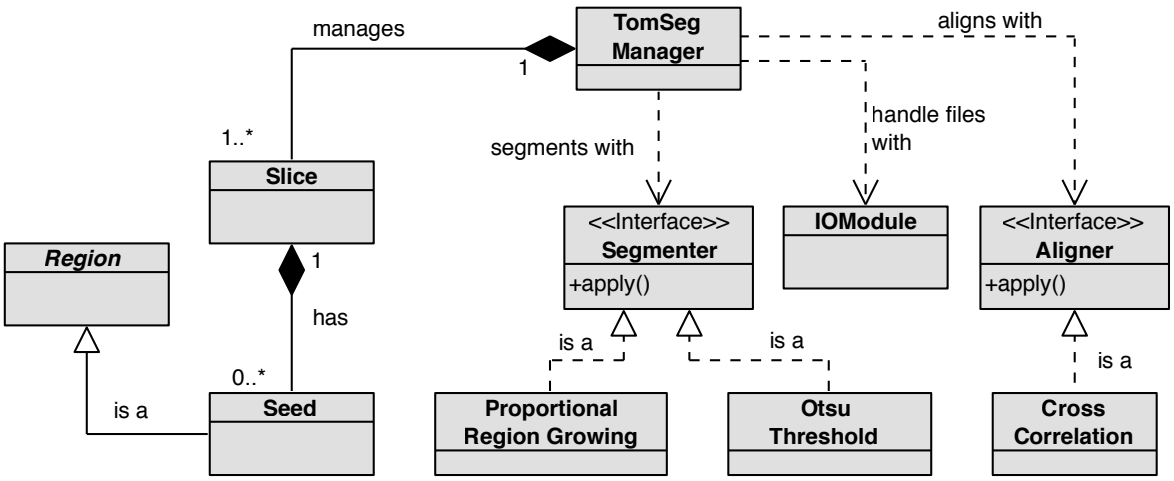


Figure 4.2.: Class diagram of main logic components of *TomSeg*

Figure 4.2 shows a diagram where the relations between the main classes are visible. There, a principal class can be distinguished: `TomSegManager`. It is the facade of all application logic, exposing an extensive list of methods. It is through these methods that the different *User Interfaces (UIs)* interact with the system. This characteristic allowed to completely separate the *business layer* from the *presentation layers*.

Additionally, the *interfaces* are even more important to the modularity of *TomSeg*. This construction allows an easy integration of new functionalities, as well as the capability of replacing the functioning of the existing ones with almost no effort. Currently, the `Aligner` and `Segmenter` interfaces encapsulate the functional operators of the align and segment procedures. To use a different algorithm it is only necessary to create a class that implements the corresponding `apply` method. From there, it is ready to be used in `TomSegManager`.

4.2 THE USER INTERFACES

As stated in Chapter 1, beyond the complexity of data analysis, what complicates even more the tomogram handling is the non-triviality associated to 3D volume visualisation.

Since tomogram handling is performed by common computer users, the interaction with the tool must be as simple as possible. This concern resulted on the *TomSeg*'s GUI, which provides an easy access to the different functionalities, allowing, at the same time, the adjustment of specific and advanced settings. Figure 4.3 presents the main UI components, and a brief description is given below.

A - CANVAS the scene where tomogram slices and results are displayed, and where parameters like the seeds and the VOI can be interactively adjusted. The workspace is slice-oriented, so only one slice is displayed at a time.

The scene magnification can either be adjusted by clicking the icons at the bottom, or by using the conventional key combinations of the current operating system.

B - 3rd DIMENSION set of components to support the navigation throughout the tomogram slices. Since thousands of them may exist, this interaction must be simple and efficient.

Jumping to the end, to the begin, or to a specific slice is trivially performed with a click. At the bottom a slider is also available to enable a fast and sequential tomogram overview, helping to review the produced results.

When the tomogram is navigated across the 3rd dimension, the whole workspace is instantaneously updated with the new slice details and results.

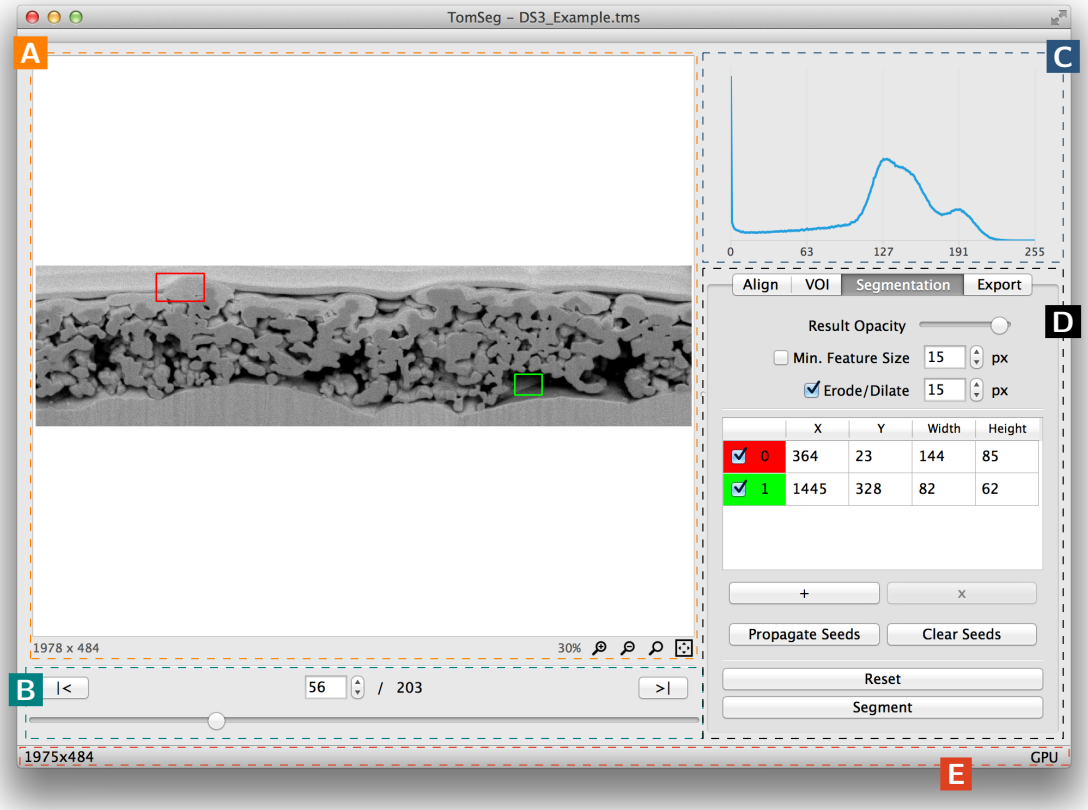


Figure 4.3.: General GUI design of *TomSeg*

C - INFO PANEL area to dynamically display the slice histogram and other specific information.

D - TOOLS interface of the different *TomSeg* modules.[2] Each one has its specific parameters. The *Canvas* area is also adapted to represent them in a user-friendly manner.

E - STATUS bar with help information: mouse position inside the *Canvas* and the active mode to compute the results (CPU or GPU).

A *Command-Line Interface (CLI)* that exposes the exact same functionalities is also available, giving the chance to integrate *TomSeg* on a script, or even as a component of a larger system.

This alternative UI is also ideal to improve the overall efficiency of both users (familiarised to a command-line interpreter) and computational systems. This minimal interface

2 Specific module UIs are presented on their respective sections.

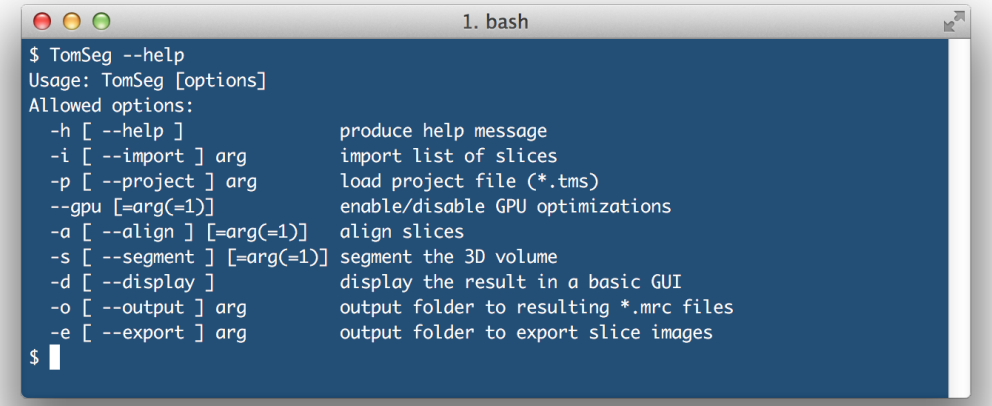


Figure 4.4.: The help message in *TomSeg*'s CLI

is also fundamental to make the execution of *TomSeg* possible on HPC environments, where GUIs are usually unavailable, or unwanted.

4.3 SLICE ALIGNMENT

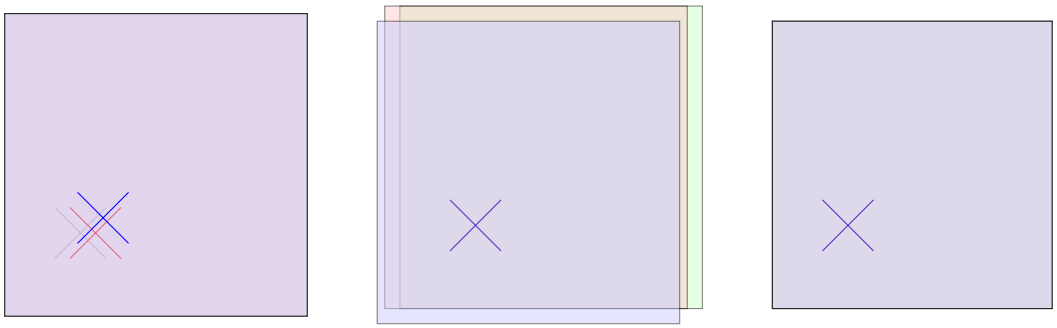
As seen in Chapter 2, before segmenting the 3D volume, the slices must be aligned to enable the accurate extraction of useful information. The cross-correlation is one of the most used and accepted methods to solve this problem, and is available in *TomSeg* in two different approaches, that the user can select:

ALIGN BY REFERENCE AREA the slices are aligned according to a feature of the image present in every slice, commonly called *marker*;

ALIGN BY PREVIOUS SLICE the slices are aligned by matching the positions of similar features between the current slice and the previous one.

In both approaches, the alignment operation is performed by computing a method based on cross-correlation to determine the shift between the slices, and then, the result is achieved

by cropping[3] them accordingly. Figure 4.5 shows an example of three misaligned slices, each containing a cross.



(a) 3 Misaligned slices (b) Shift computed (c) Cropped slices

Figure 4.5.: Alignment procedure of *TomSeg*

As can be seen by the example, the size of the slices may not remain constant during a *TomSeg* instance. The resultant aligned volume is the intersection between the shifted slices. This is important so that all the slices are the same size throughout the 3rd dimension.

Among these different steps, the alignment time is defined by the complexity of the procedure that finds the shift distances between the slices. To improve both efficiency and quality of the results, users can provide additional information about the maximum displacement between two consecutive slices (Figure 4.6). In most cases, this allows *TomSeg* to focus the search on a much smaller region, preventing, at the same time, disproportionate results.

4.3.1 *Align by Reference Area*

One alternative to align volume slices in *TomSeg* is by selecting a reference area, which is used as a reference to align all slices. As stated before, this method is appropriate on datasets where a same feature is constantly present along the 3rd dimension.

The selection of the *reference area* on the GUI can be done by drawing interactively an area over a slice, directly in the *Canvas*, or by adjusting the values manually on the *Tool* panel.

3 Slice alignment is a destructive process. This concept is crucial for loading and saving *TomSeg* project files (more details in Section 4.6).

The selection remains visible on the *Canvas* until the user decides to *Align* the volume or to *Reset* the selection, as visible in Figure 4.6.

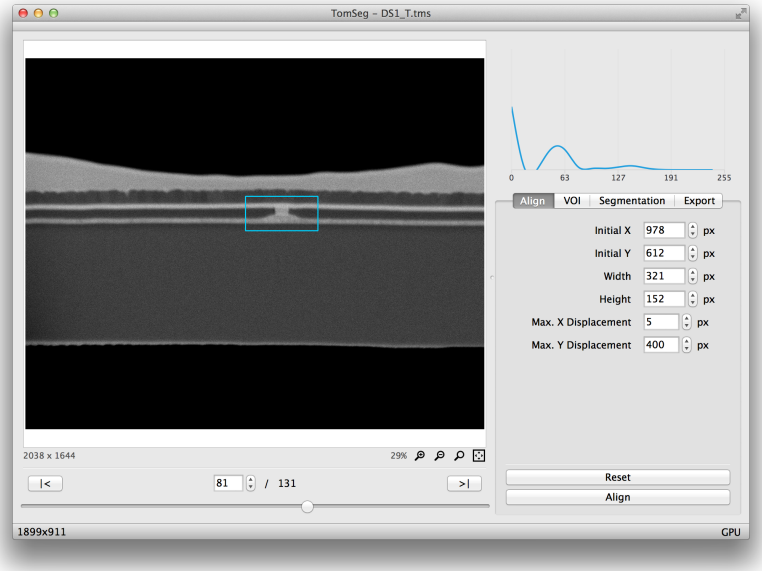


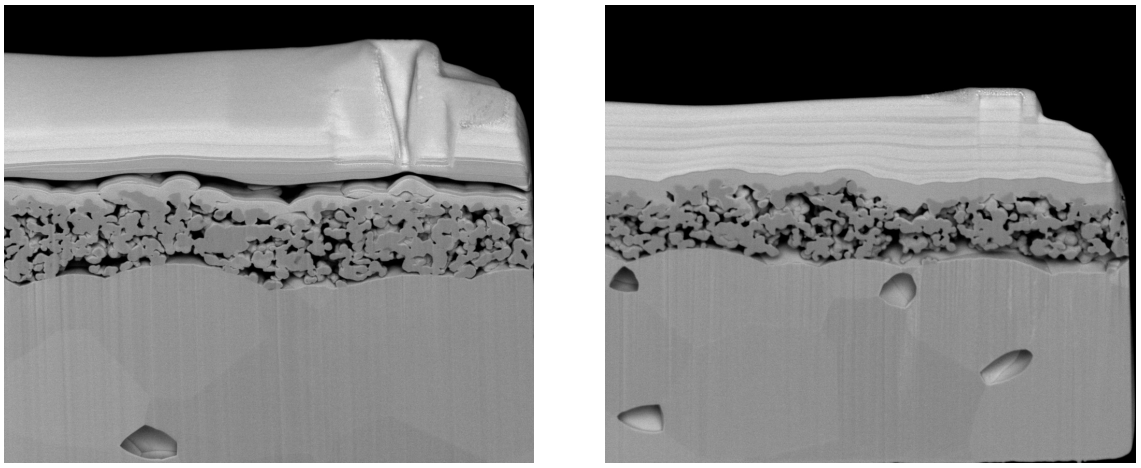
Figure 4.6.: Volume alignment using a *reference area*

As seen in Chapter 3, since the alignment task is quite hard, any attempt to make a better use of the computational resources is beneficial. In this case, the multiple *cores* available on the common machines can be very helpful to accelerate the process. After selecting the area, the search can be performed concurrently in the different slices without any dependence. The performance results of this approach are presented and discussed in Chapter 5.

4.3.2 *Align by Previous Slice*

Another approach to align volumes on *TomSeg* is by inspecting and aligning the slices in sequence. This is useful to align volumes whose slices vary a lot on the 3rd dimension and a reference marker is not available. Figure 4.7 shows two slices from a tomogram of a thin-film component of a fuel-cell device, and represents a good example where aligning by the previous slice produces best results.

In contrast to alignment by a *reference area*, this alignment procedure is based on the entire images. This means higher complexity and, therefore, more time to align. Furthermore, it is



(a) Slice #1 (b) Slice #192

Figure 4.7.: Tomogram with no reference markers

not trivial to align the different slices in parallel: the correct alignment of a slice n depends on the correct alignment of the slice $n-1$. This characteristic represents a flow dependency known as *Read After Write (RAW)*: it is only possible to align the next slice after aligning the current.

However, *TomSeg* supports parallelism on this task. This problem was settled by dividing the alignment process into three different steps: compute the relative displacements, accumulate them, and crop the slices.

Instead of computing the displacement between a pair of slices and align immediately, *TomSeg* computes the relative displacements between each pair of slices, and accumulates them in the end. Only then, with the absolute deviations calculated, the effective align take place, by cropping the slices accordingly.

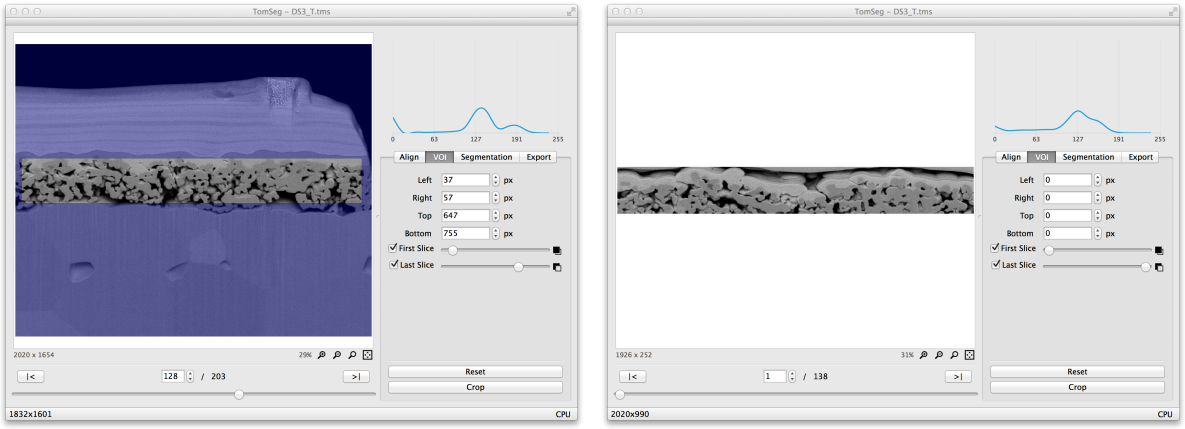
It is important to note that the RAW dependency did not vanish. This solution only restricted its scope to a smaller execution section: the accumulation of the relative displacements. However, this was enough to enable the parallel and independent execution of the cross-correlation method, which is the main bottleneck. With this technique, the alignment of the volume presented in Figure 4.7 (203 slices with 2048×1768 pixels each) can be performed in 2.8 seconds, instead of 54 seconds (details in Chapter 5).

4.4 VOI SELECTION

Due to the imaging method and to the characteristics of the sample, tomograms may contain data about regions that are not relevant to the analysis. Thus, it becomes fundamental to select the important information and work with it from a given moment. As an example, the volume presented in Figure 4.7 reflects this exact importance.

Still focusing on these pre-segmentation requirements, *TomSeg* provides a tool with which the user can crop the volume by selecting a Volume of Interest.

In *TomSeg*, the selection of the 3D sub-volume is done by picking the initial and final slices (3^{rd} dimension) and by adjusting the interesting region on the *Canvas* area. The area of interest must be constant throughout the slices, so the volume must be previously aligned. The corresponding 3D selection can be always inspected by navigating through the slices using the *3^{rd} Dimension* controls.



(a) Selection (b) Result

Figure 4.8.: Defining a VOI in *TomSeg*

Figure 4.8 shows the previous tomogram being cropped using *TomSeg* tools. After aligning the volume the user decided to remove redundant slices from the beginning and the end, and to focus the analysis on the porous material. The resulting volume is a smaller parallelepiped of size $1926 \times 252 \times 138$, containing only the selected information.

When the volume is cropped, the discarded slices and the deleted pixels are removed from the main memory. However, this becomes a problem when saving *TomSeg* project files. As tomogram sizes may take gigabytes of space, to replicate the edited slices would

represent a waste of memory that users would not tolerate. Instead, each slice object contains not only the current image, but also additional information to specify the region it represents relatively to the original image. Every time a `crop` is applied to a slice, this value is updated. In this way, once a project is loaded, the edited slices are recovered by applying the corresponding transformation to the original images.

Since the alignment operation is achieved by cropping the slices appropriately, once a project is loaded, the volumes will be also aligned. It is important to note that this alignment correction does not involve cross-correlation methods: it is a side effect of applying the *Region of Interest (ROI)* to the original images.

4.5 A Novel 3D Segmentation Process

When the tomogram is correctly aligned and its Volume of Interest defined, the segmentation can finally take place.

During the design of this tool several different approaches were tested. However, they all pointed to the same difficulty: with the available computer vision techniques it is very hard to segment an experimental tomogram in just one step. Thus, both to segment the volumes correctly, and to take advantage of the hardware characteristics, the *TomSeg*'s main segmentation procedure is divided in several phases.

As mentioned in Chapter 3, another important constraint imposed to *TomSeg* is to orient the segmentation to a single slice, i.e., the algorithm to segment a slice should avoid accessing voxels on another. The segmentation on the 3^{rd} dimension should be performed using a different, lightweight, algorithm, re-segmenting only the areas that differ between two consecutive slices.

4.5.1 *Seeds*

While developing and testing segmentation procedures based on thresholding and boundary detection techniques, several adjustments had to be performed to the algorithms, according to dataset specific characteristics (Section 2.4.2).

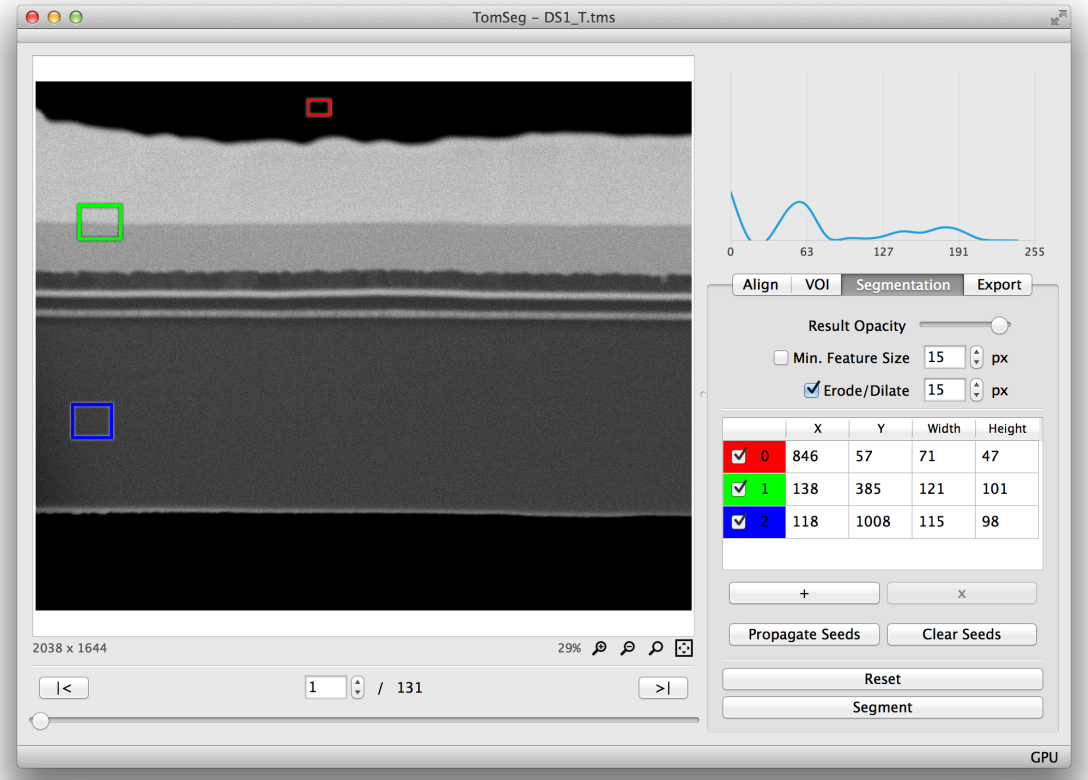


Figure 4.9.: Defining slice *seeds* on *TomSeg*'s segmentation tool

To turn the segmentation procedure more natural and generic, *TomSeg* tried to approximate its automatic approach to the versatile human methods of defining concepts such as the visual continuity of an image region. Thus, exploring region growing based procedures (more details in Chapter 2) became the main focus.

For *TomSeg* to distinguish the different *voxel* classes present on the tomogram, the user must define different *seed* regions. Each *seed* represents a class, so it must contain a representative set of *voxels*. In the context of *TomSeg*, a slice containing *seeds* is a *seeded slice*.

Figure 4.9 shows the main GUI of *TomSeg* after the user has defined the *seeds* on the first slice. The tomogram is ready to be segmented in those three classes. However, to take advantage of multithreading and to improve the quality of final results, more *seeded slices* must exist (details in Sections 3.3.2 and 4.5.2). This can be accomplished manually, by redrawing the *seed* areas throughout different slices, or by using a fully automated feature of *TomSeg*.

4.5.2 *Seed Propagation*

Since *TomSeg* segmentation procedures have the capability of splitting the volume at each *seeded slice* and processing the sub-volumes independently, users may want to have several *seeded slices*. Only so they can take advantage of their multi-core CPU-chips to accelerate the segmentation process. In addition, since segmentation of non-seeded slices is based on the results of fully-segmenting a *seeded slice*, the existence of several *seeded slices* attenuates errors propagation across the 3rd dimension.

To exempt the user from re-introducing the *seeds* in several slices, *TomSeg* provides an automatic *seed propagation tool*. The `SeedPropagator` replicates the *seeds* from a reference slice across the volume with a stride defined by the user.

On a non-*seeded* slice, the first approach is to place the new *seed* in the same 2D coordinates. However, if *TomSeg* detects[4] that the new region does not represent the same class as the one on the initial *seeded slice*, it tries to place it on another valid, nearby position.

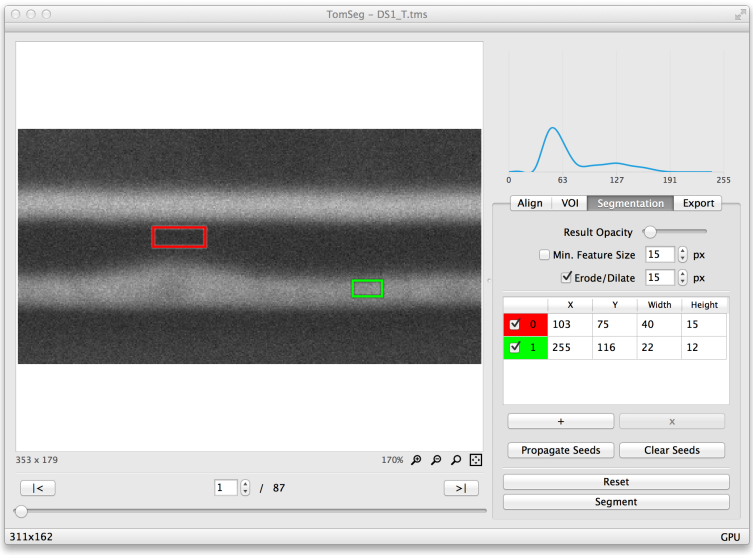
Figure 4.10 shows a basic example of the *seed propagation* feature with the two possible scenarios. Image 4.10a shows two *seeds* defined by the user, which were then propagated using the automatic feature of *TomSeg*. However, by navigating the slices on the 3rd dimension it is seen that the two parallel pieces are connected, and one of the propagated *seeds* can not be created at the same position (4.10b). In the last tomogram slices, the pieces are not connected again, allowing new *seeds* to be created directly at the reference positions (4.10c).

4.5.3 *Proportional Region Growing*

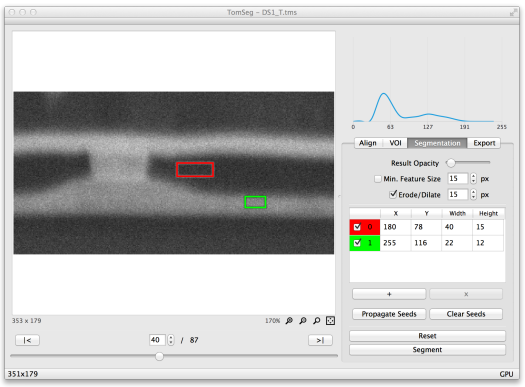
To segment volumes, *TomSeg* starts by segmenting their first *seeded slice* with a procedure called *Proportional Region Growing*. That result is then used as a reference to segment the following non-*seeded* slices more quickly. It must be as accurate as possible.

In this procedure, four main phases are distinguishable (Figure 4.11). Mostly based on region growing methods, morphological filters, and custom tailored algorithms, their aim is to:

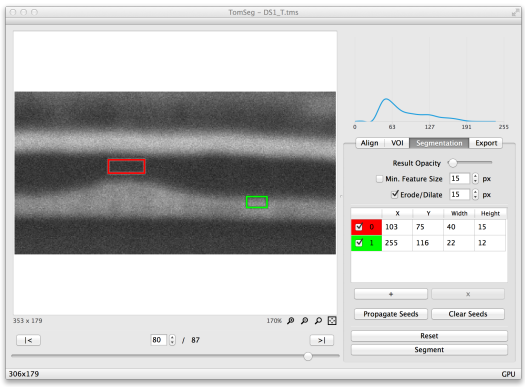
4 If $|\bar{x}_{newRegion} - \bar{x}_{originalSeed}| \geq 2 \times \sigma_{originalSeed}$, *TomSeg* discards the region.



(a) User defined



(b) Propagated (*1* shifted)



(c) Propagated (direct)

Figure 4.10.: *Seed propagation* in *TomSeg*

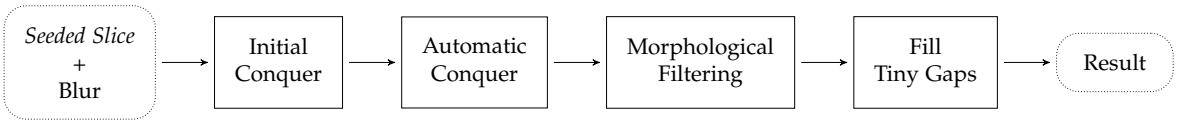


Figure 4.11.: The phases of *Proportional Region Growing*

INITIAL CONQUER expand the initial *seeds*;

AUTOMATIC CONQUER segment unvisited areas;

MORPHOLOGICAL FILTERING remove segmentation imperfections;

FILL TINY GAPS close tiny unsegmented points.

To reduce image noise, the *seeded slice* is blurred before applying the *Proportional Region Growing* procedure.

Since region growing based methods are the main key to *TomSeg* segmentation, they have been carefully designed and tuned. As discussed below, on different segmentation phases, different *assimilation criteria* are used. To improve modularity, the main region growing skeleton, common to all these phases, is implemented as a high-order function, taking the assimilation criteria function and its parameters as argument. This way, if the skeleton is improved, all methods based on region growing will also be.

Initial Conquer

Initial Conquer is the first phase of *Proportional Region Growing* procedure. Its main aim is to expand the initial *seed* regions as much as possible, always maintaining the segmented regions continuous. *Initial Conquer* achieves this by combining both histogram and region growing techniques.

The algorithm starts to analyse the mean ($\bar{x}$) and standard deviation (σ) values of the *seed* areas from the slice intensities. The *seeds* are then sorted by crescent averages and the histogram is divided in *proportional* chunks, i.e., the amount of grey-levels a *seed* may conquer (instead of another) is proportional to the difference between the standard deviation values of the two. Thus, a class with a higher tone variance will be able to conquer a wider set of *voxels* than a more homogeneous one.

Equations 2 and 3 define the histogram grey-levels in which the chunks must begin and end, respectively.

$$ChunkStart_i = \begin{cases} 0, & i = 1 \\ \bar{x}_{i-1} + \sigma'_{i-1} \times \dfrac{\bar{x}_i - \bar{x}_{i-1}}{\sigma'_i + \sigma'_{i-1}}, & 1 < i \leq N \end{cases} \quad (2)$$

$$ChunkEnd_i = \begin{cases} ChunkStart_{i+1} - 1, & 1 \leq i < N \\ 255, & i = N \end{cases} \quad (3)$$

i: *seed* number, sorted by average

N: number of *seeds*/histogram chunks

σ': $max(1,\ \sigma)$

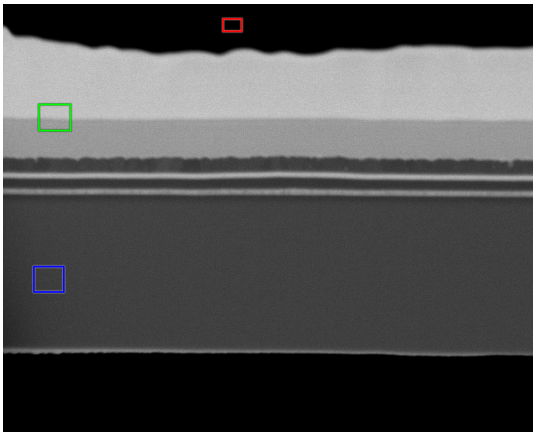


Figure 4.12.: *Seeded slice*

i		$\bar{x}$	σ	*cStart*	*cEnd*
1	red	0	0	0	6
2	blue	61	8	7	92
3	green	176	21.12	93	255

Table 4.1.: Sorted seeds data

To make this concept clearer, Figure 4.12 shows a tomogram slice with three *seeds* defined. The statistical parameters of each *seed* is summarised in Table 4.1: the average and standard deviation, and the results of Equations 2 and 3, defining the proportional chunks.

Figure 4.13 presents all data presented above graphically. The solid lines are the *seeds*' mean values, and the background patterns their respective chunks. By analysing the figure, it becomes clear that *seeds* with a higher standard deviation get more histogram area. As an example, the red *seed*, which has a null standard deviation, only got 7 grey-levels to its right. On the other hand, its neighbour (blue *seed*) got 54 levels. The same occurs between

blue and green *seeds*: since the green *seed* has a higher standard deviation, it gets more grey-levels than the blue (83 *vs.* 32 grey-levels).

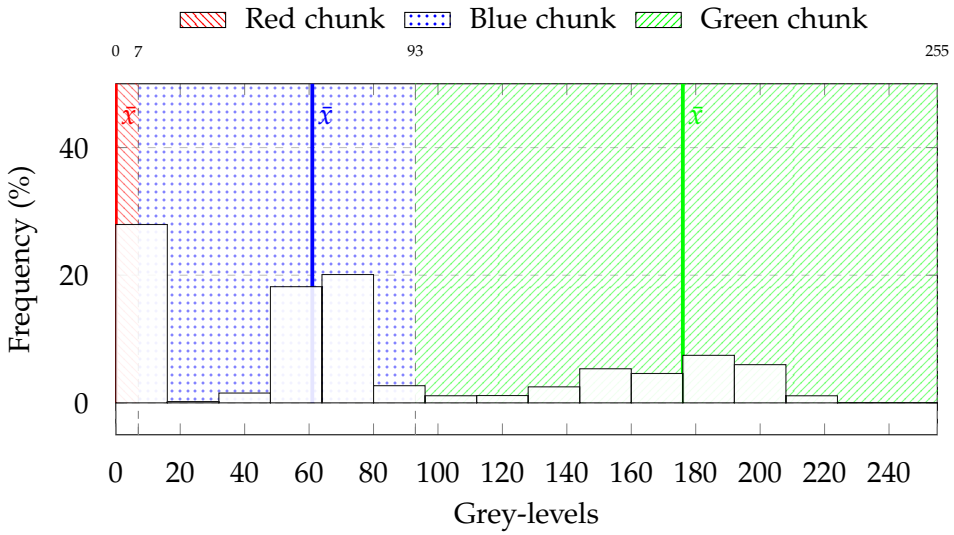
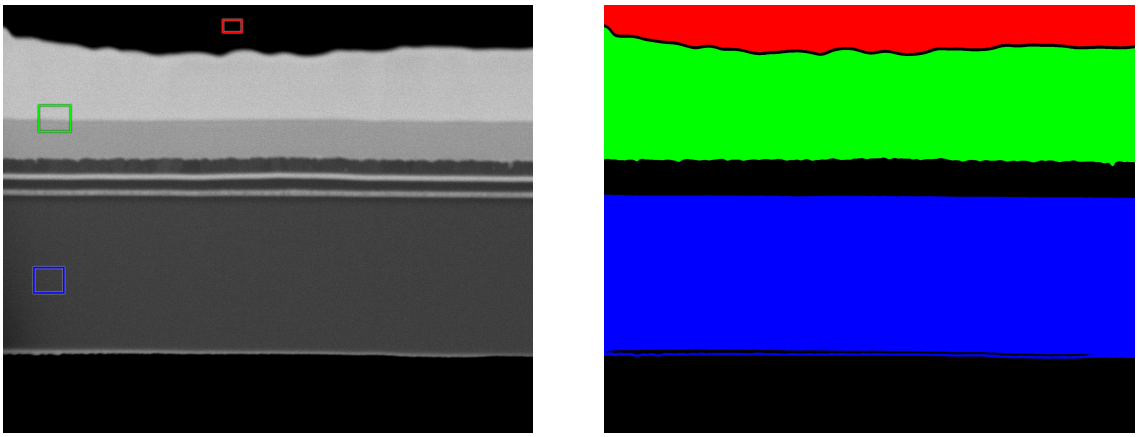


Figure 4.13.: Histogram with chunks represented

With the histogram divided, *Initial Conquer* can now start expanding the initial *seeds*. As expected, the custom *assimilation criteria* is based on the computed chunks: at each iteration all *voxels* at the border of the growing region are examined, appending only those whose value belongs to the respective chunk. This process is repeated until no *voxel* is appended.



(a) Seeded slice (b) Partial result

Figure 4.14.: Partial result obtained with *Initial Conquer* phase

The partial segmentation resulting from this process can be seen side by side with the original *seeded slice* in Figure 4.14, where the black colour on the result image represents unsegmented *voxels*. This allows to verify that almost no *voxel* was incorrectly classified.

Automatic Conquer

More than 27% of the slice is not segmented yet: the regions have stopped growing on the boundaries with other classes. To continue the segmentation process, *TomSeg*'s *Proportional Region Growing* moves to its next phase — *Automatic Conquer*.

This phase assumes that if new *seeds* can be automatically found and mapped to an existing class/*seed*, then, the unsegmented area can be explored and segmented by other region growing based methods.

To automatically find new *seeds*, the algorithm takes advantage of an user defined parameter which defines the *Minimum Feature Size*. This value defines the size of the tiniest interesting image characteristic. Thus, the algorithm begins to search the partial segmentation result for unsegmented areas larger than that value.

When a larger area is found, the procedure tries to detect if that region is a boundary between two (or more) classes. This is done by computing the area's *centre of mass*. If it does not match its physical centre ($\pm 5\%$), the area is not converted into a new *seed*.

Once a valid *seed* is found, two hypothesis are considered:

SIMILAR SEED the new automatic *seed* has average and standard deviation values very similar to those of an original *seed*. Thus, the class of the new automatic *seed* is assigned directly.

Given the similarities between the new *seed* and the original, the *assimilation criteria* used to explore the new area is the same used in the *Initial Conquer* phase (based on the same chunks).

DEGENERATED SEED the statistic values of the new *seed* do not match with any original *seed*. In those circumstances, a hypothesis evaluation is done. This hypothesis analysis combines statistical information (about the mean and the standard deviation values) and spacial information (about the physical distance between the new area and the already segmented *voxels*) to calculate a percentage grade that dictates the chances of

the new area to belong to each of the different classes. Based on this values, the best graded class is assigned to the new *seed*.

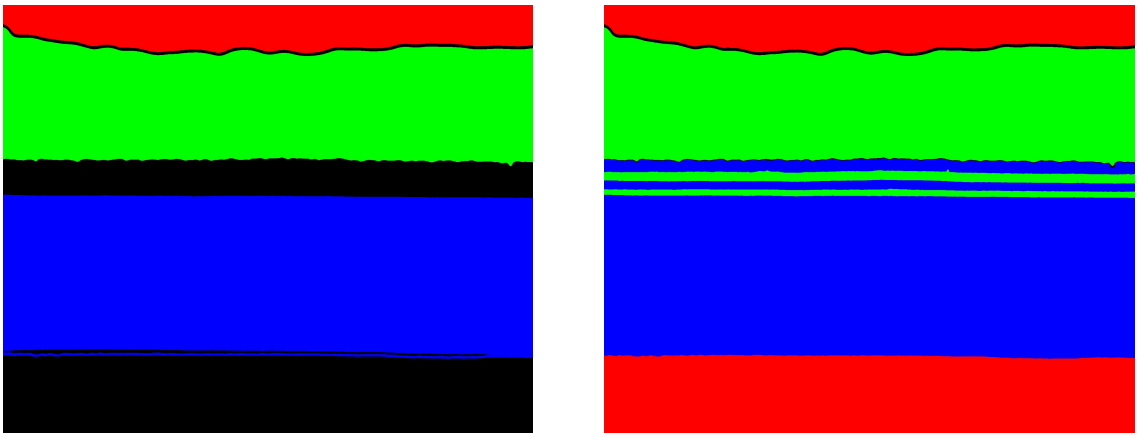
As the *seed* is *degenerated*, the same *assimilation criteria* should not be used. Instead, a new one was designed, based directly on the new *seed*'s mean and standard deviation values. This *assimilation criteria* is also configurable with a relax factor (Eq. 4), which lets the region to grow faster. However, high relax values may reduce the quality of the final segmentation results.

$$RelaxedInterval = [\bar{x}_{seed} - r \cdot \sigma_{seed},\ \bar{x}_{seed} + r \cdot \sigma_{seed}] \tag{4}$$

The appropriate *assimilation criteria* is then defined by: at each iteration all *voxels* at the border of the growing region are examined, appending only those whose value belongs to the relaxed interval. This process is repeated until no *voxel* is appended.

When no more *seeds* can be found, i.e., there are no more valid areas larger than the *Minimum Feature Size* to segment, this phase ends.

Starting from the previous phase results, at the end of the *Automatic Conquer* (Figure 4.14), using a relax factor of 2 ($r = 2$), only less than 1% of its *voxels* are unsegmented. All the remaining were successfully classified, as visible on Figure 4.15. Table 4.2 synthesises the whole functioning of this phase, presenting each of the six automatic *seeds* along with its relevant metrics and results.



(a) *Initial Conquer* partial result (b) *Automatic Conquer* result

Figure 4.15.: Partial result obtained with *Automatic Conquer* phase

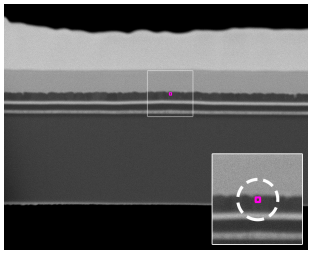
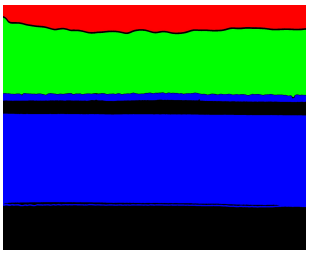
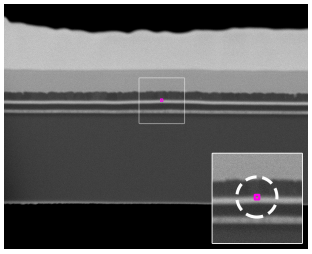
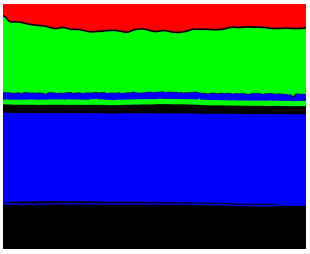
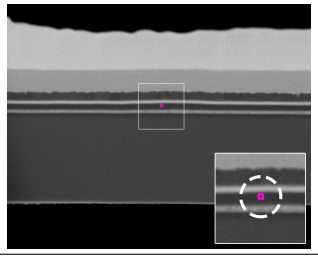
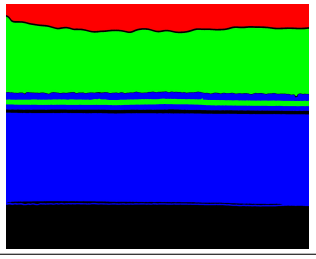
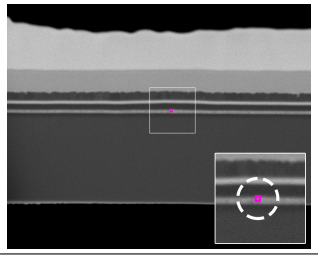
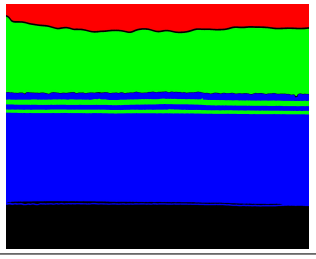
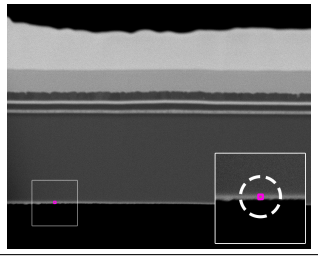
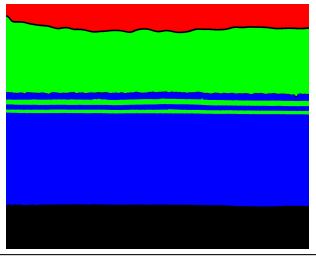
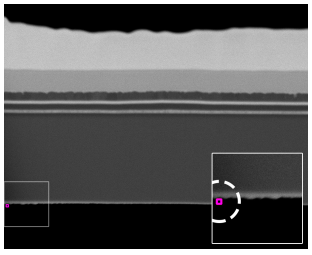
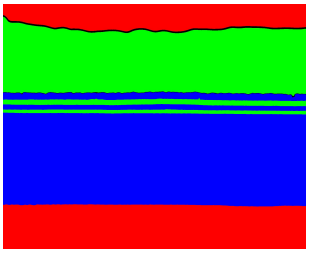
#	Automatic *Seed*	Type	Assigned Class	Region Growing Result
1		Degenerated *($\bar{x}$:64, σ:7.3)*	blue *(grade: 90%)*	
2		Degenerated *($\bar{x}$:126, σ:30.6)*	green *(grade: 91%)*	
3		Similar *($\bar{x}$:54, σ:7.9)*	blue	
4		Degenerated *($\bar{x}$:124, σ:18)*	green *(grade: 88%)*	
5		Degenerated *($\bar{x}$:115, σ:14.1)*	blue *(grade: 90%)*	
6		Similar *($\bar{x}$:0, σ:0)*	red	

Table 4.2.: Seeds found during *Automatic Conquer*, and respective results

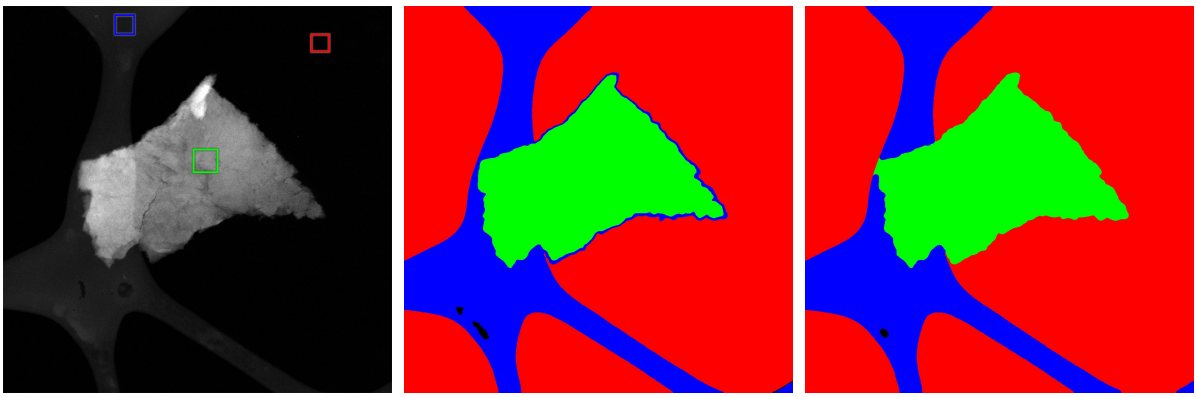
Morphological Filtering

During the execution of the last two phases some segmentation errors may have been introduced: the proportional chunks did not reflect the real histogram distribution (especially at class boundaries); a wrong decision about which class a new *seed* belongs; etc. To correct them, *TomSeg*'s segmentation procedure provides an optional *Morphological Filtering* phase.

Morphological Filtering main goal is to correct small isolated segments that may be dispersed across the result. To accomplish this, *TomSeg* makes use of an *open* operator (equivalent to $dilate \circ erode$).

Since *morphological filters* replace a value by another, a coherent hierarchy between the values must exist. In *TomSeg* such hierarchy is defined by users when seeding the slice. The order in which the seeds are created (if not sorted after) determines the relevance of their respective class.[5] Lower *seeds*, i.e., with lower *ids*, will be "stronger" than higher ones.

Figure 4.16 shows a segmentation result with some imperfections, specially on the boundary area between the sample and the background. As the *red* class must conquer the *blue* imperfections, its seed must be defined first. Then, just by applying the *Morphological Filtering*, *TomSeg* gets the results corrected — as $red > blue$, after the applying the *open* operator, the *blue* boundary between the sample and the background was conquered by *red*.



(a) Seeded slice (b) *Automatic Conquer* result (c) Result

Figure 4.16.: Results of *Morphological Filtering*

5 Analogously, when using the CLI, the relevance of each class is determined by the order in which the *seeds* are represented on the JSON project file (more details in Section 4.6).

Fill Tiny Gaps

In both results of Figures 4.15 and 4.16, segmentation is not totally completed. On the latter, 0.88% of its *voxels* are not segmented. This number is even lower on the former, with unsegmented *voxels* representing less than 0.04%. However, this number must be zero: an ideal segmentation algorithm classifies all voxels as one of the specified classes.

This last phase ensures that no region is left unsegmented. Inspired on region growing methods, this algorithm starts by searching an unsegmented point. Once one is found, it begins to expand through its unsegmented neighbours, searching for segmented *voxels*. At the end of this search, a list of accessible classes is created. Then, based on a direct comparison between the mean and standard deviation values, the most similar class is selected to append those unsegmented points. The process is repeated until the slice is fully segmented.

The final segmentation result of the two given examples is visible in the Figure 4.17. The results are very similar to those obtained by the end of *Morphological Filtering*, however, now, all *voxels* are classified as one of the defined classes.

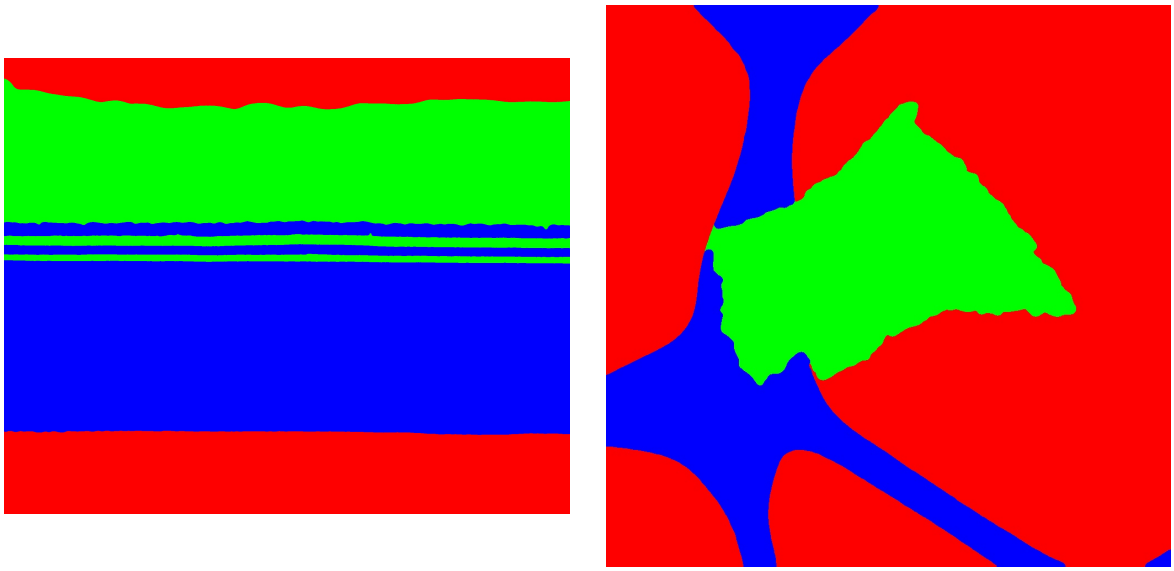


Figure 4.17.: *Proportional Region Growing* final results

4.5.4 *Differences Between Slices*

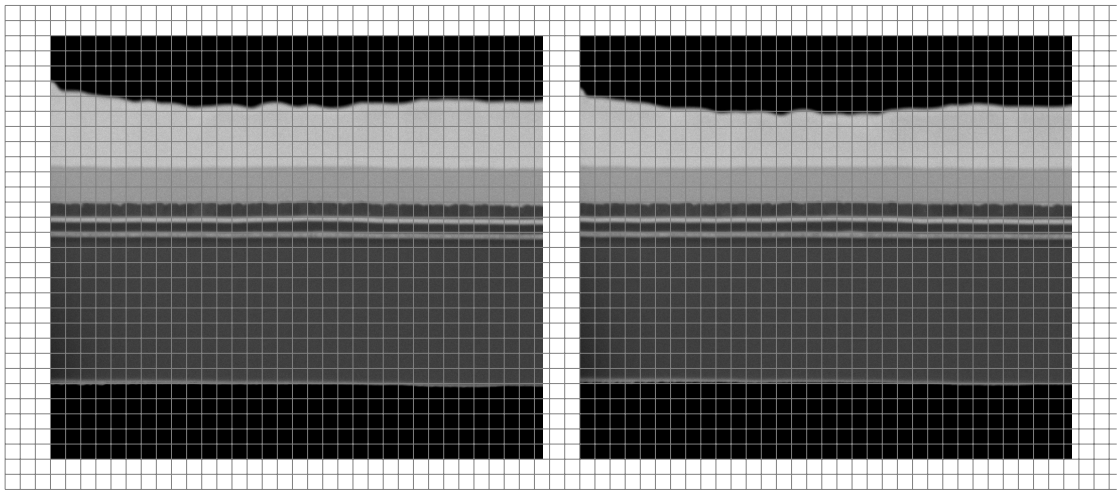
After segmenting the *seeded slices*, it is fundamental to process the remaining non-*seeded* slices of the volume. Depending on the dataset and its configuration, this task may be responsible to segment hundreds of slices. To minimise the impact of this operation, some assumptions can be made based on the characteristics of the volumes.

Given that materials have 3D continuity, the neighbouring tomogram slices have a lot of information in common. Thus, *TomSeg* assumes that the segmentation result of slice $n+1$ will be equal to the results obtained to slice n, except in the areas where main differences are distinguishable.

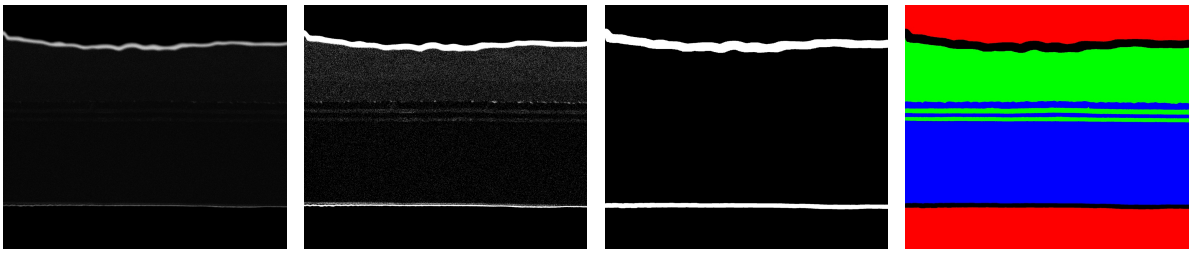
To detect differences between slices, *TomSeg* subtracts the images and performs some basic filtering to distinguish structural differences from random noise. At these different regions, the reference segmentation is reset and re-segmented by *Proportional Region Growing*.

The process of computing the differences between two slices is shown in the Figure 4.18, which presents two neighbour slices of a known tomogram. If a careful analysis is made, it can be noticed that the images differ slightly at the top and bottom. A re-segmentation is only required at those regions. The remaining slice regions were directly segmented by the reference (copy of the the previous slice results).

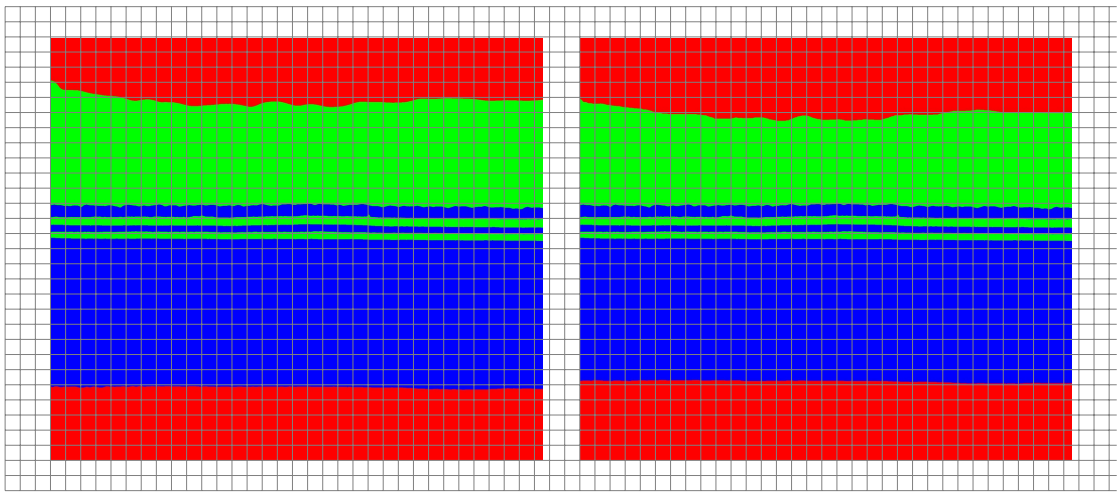
Although this method works well on accelerating the segmentation process, it has some limitations. Since segmentation of non-*seeded* slices is deeply based on previous results, this method can easily propagate errors throughout the volume. To improve both performance and segmentation quality, it is important to have several *seeded slices*. Those are segmented independently, stopping the propagation.



(a) Slices 1 & 2



(b) $|slice_1 - slice_2|$ (c) Threshold (d) Morph. open (e) Seg. differences



(f) Segmentation results of slices 1 & 2

Figure 4.18.: Differences between two consecutive slices and final results

4.6 TOMSEG PROJECT FILES

Handling tomograms is sometimes a complex and time-consuming job. Being fully aware of this, *TomSeg* allows saving and loading project files, at any time. This functionality, in addition to allowing users to pause working whenever they want, was designed to be efficient in resources usage, and to expand the tool capabilities.

Based on the *JavaScript Object Notation (JSON)* format and on the tool's lightweight mechanism of representing edited slices, the *TomSeg* files (`.tms`) describe the projects in very small files, not replicating any image data.

For project details to persist, `tms` files organise the data in a hierarchy very similar to *TomSeg* classes: a root object stores global parameters and a list of `slices`; in turn, each `slice` is characterised by the path to the original image, its ROI[6], and a list of `seeds`. Listing 4.1 presents a pseudo-JSON snippet describing the structure of these documents.

```
{
  minimumFeatureSize: int,
  morphologicalSize: int,
  useGPU: boolean,
  xLen: int, yLen: int, zLen: int,
  slices: [{
      path: string,
      roi: {
          x: int, y: int,
          width: int, height: int
      },
      seeds: [{
          id: int,
          x: int, y: int,
          width: int, height: int
      }]
  }]
}
```

Listing 4.1: Structure of the *tms* file format

Being textual and readable by humans, these files can be easily edited in both graphical and command-line general purpose applications. This also allows *TomSeg* to be easily integrated in more complex *scripts* or applications, enabling them to configure the tool au-

6 The ROI field allows *TomSeg* to recover the edited slice from the original image.

tomatically, just by generating these files. On the other hand, *TomSeg* file format can also be used to enable collaborative work, or to port the project to another machine.

Additionally, project files are also fundamental when the user wants to switch between user interfaces. In more graphical/interactive tasks, such as VOI and *seed* selection, most users prefer to use the GUI. However, to some users and environments, the usage of the CLI is fundamental when performing compute intense tasks. The transition between the two can be done by saving the project on the former, and then loading it back on the latter.

4.7 Integration with Visualisation Tools

After segmenting a tomogram into a simplified volume of classes, scientists need to make qualitative and quantitative analysis of each phase. As the Graphical User Interface of *TomSeg* are slice-oriented, reasoning about concepts like the 3D appearance of the volume structure becomes hard. Thus, *TomSeg* allows one to export the alignment and segmentation results in formats that are widely accepted by the main visualisation software packages.

Figure 4.19 shows the GUI of the exporting tool. If *TomSeg* is used only to align and/or select a Volume of Interest, the edited results can be directly exported as a stack of images. If *TomSeg* proceeds to segmentation, it may export each *voxel* class as a different 3D volume.

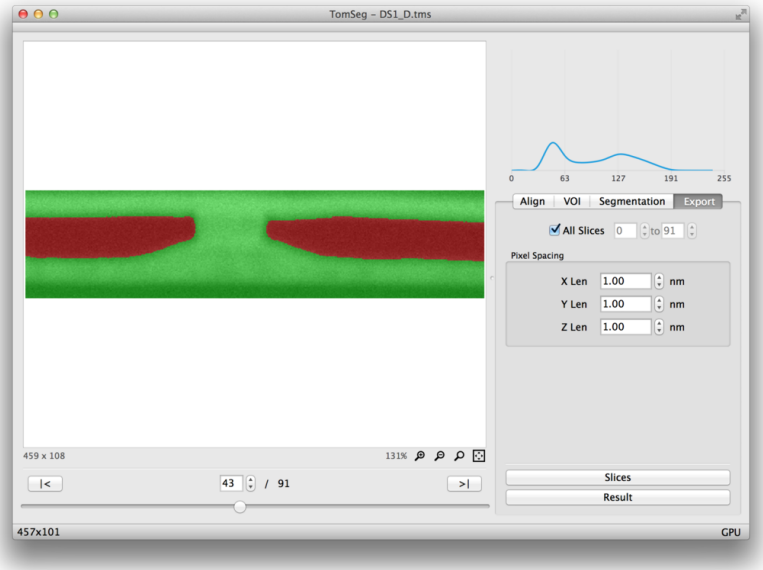


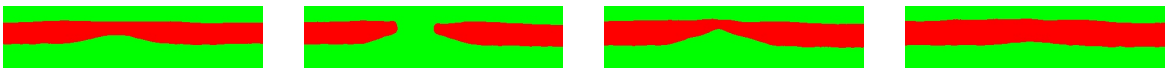
Figure 4.19.: *TomSeg*'s exporting tool

4.7.1 *Exporting MRC Files*

The binary file format proposed by the *Medical Research Council* is one of the most common formats to represent volumetric data (more details in Appendix A).

Based on the scheme presented in Figure A.1, the `IOModule` of *TomSeg* compiles the segmentation results of each class into a different file, i.e., the segmentation result of a tomogram with N classes is exported as N distinct `mrc` files. The results are exported in separate files to allow users to load only the interesting classes in the visualisation package.

To show the relevance of such visualisation packages, four segmented slices of the previous tomogram (Figure 4.19) are displayed in Figure 4.20. By inspecting the slices on the 3^{rd} dimension, experienced users can conceptualise a 3D image of the two materials getting connected. However, even for these users, it is hard to figure out the geometric appearance of such connection just by inspecting the segmentation results slice-by-slice.



(a) Slice 0 (b) Slice 30 (c) Slice 60 (d) Slice 90

Figure 4.20.: Segmentation results of four slices

4.7.2 *Visualisation with Chimera*

To overcome this difficulty, segmentation results should be imported to a visualisation package. *Chimera* is the recommended to be used along with *TomSeg*. It is a lightweight and complete visualisation software, capable of reading and displaying multiple `mrc` files. These files can either be visualised one-by-one, or simultaneously.

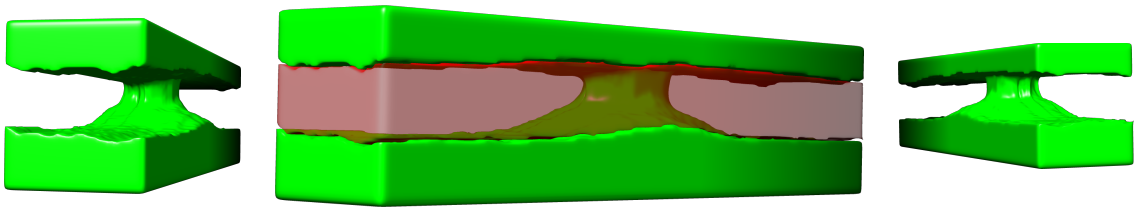
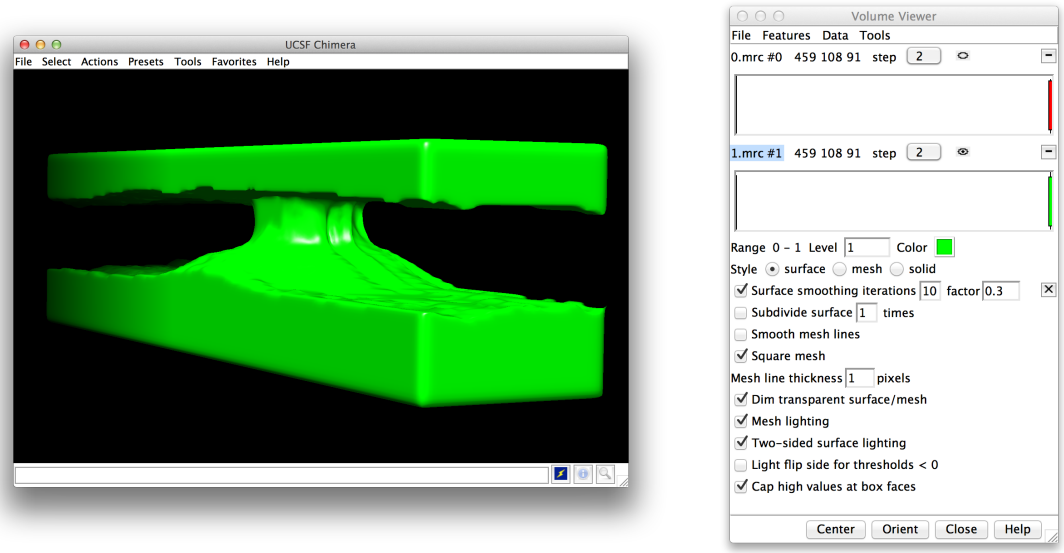


Figure 4.21.: Segmentation results visualised in *Chimera*

Figure 4.21 shows three snapshots of the rendered volume using *Chimera*. With them, and by moving interactively across the volume, the geometric aspect of the previous sample becomes clear: two parallel bars get connected by a smooth cylindrical piece (more details of the sample in Section 5.1.1).

On experiments in which an extremely precise qualitative analysis is not fundamental, it is highly recommended to enable the *surface smoothing* feature for a better visualisation experience. The volumes presented in the snapshots were smoothed with a 0.3 factor (Figure 4.22b).



(a) Volume panel (b) Settings

Figure 4.22.: *Chimera*'s main GUI components

5

VALIDATION AND PERFORMANCE ANALYSIS

To ensure that the produced results are correct and acceptable, several datasets were provided by the INL. To validate and tune the performance of *TomSeg*, two different experimental environments were targeted: a laptop and a compute server. The performance results that are analysed address scalability issues in both multi-core based platforms, measuring execution times for each of the two more computationally intensive procedures: the alignment and the segmentation. The code bottlenecks that required an investment to improve the code performance are also displayed with execution graphs.

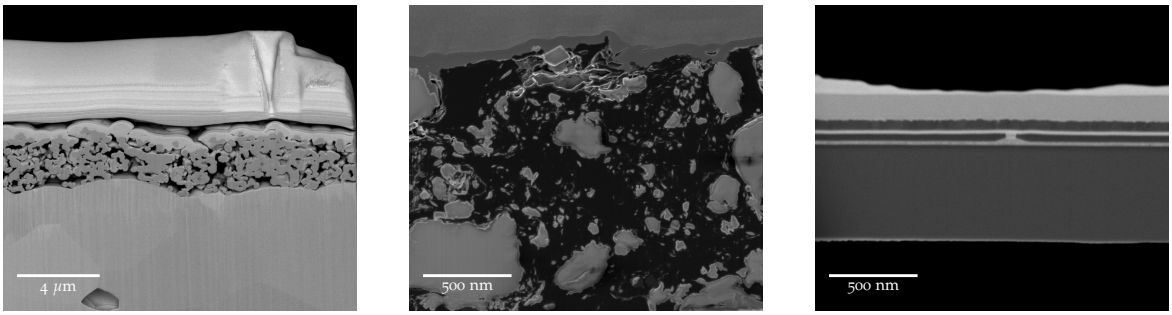
5.1 THE TESTBED ENVIRONMENT

5.1.1 *The Datasets*

To validate the *TomSeg* outcomes and to provide a comparative evaluation of their performance, the presented results focused only on a small set of 3 tomograms. The selection was based on image characteristics, size, number of slices, and number of classes. Their technical details are summarised in Table 5.1 and their description follows with Figure 5.1.

Dataset	#Slices	Width	Height	#Classes	Size
DS1	203	2048	1768	2	729 MiB
DS2	600	2048	1768	2	2.24 GiB
DS3	131	2048	1768	3	501 MiB

Table 5.1.: Details of the datasets



(a) DS1 (b) DS2 (c) DS3

Figure 5.1.: Random slices from the selected datasets

DS1 is a tomogram of a thin-film component, present on fuel-cell devices. In this sort of problems the common interesting characteristics are the pores connectivity, their total volume, and surface area.

DS2 is a tomogram of an under water sedimentary rock — *shale*. The goal of the experiment is to determine its volumetric fraction that can be processed to generate fuel.

In the context of *TomSeg*, it is mainly used to test the capacity of handling huge and complex datasets. It has 600 slices, each with more than 3 MiB, representing a total data size of 2.24 GiB.

DS3 is a 131-slice tomogram with three main classes. This dataset shows a *Magnetic Tunnel Junction (MTJ)*: two ferromagnets separated by a thin insulator (≈ 100nm) To collect images from that feature, more materials were added when building the sample. As those materials are irrelevant for the study, they can be discarded before segmentation begins.

Requiring aligning, selecting a VOI, segmenting, and visualising the results in the 3D space, this tomogram is used to validate the full workflow of *TomSeg*.

5.1.2 *The Platforms*

The performance analysis of *TomSeg* targeted two different experimental environments: a laptop and a compute server. The former aims the execution on personal devices, while the latter tests the efficiency of the solution on an environment similar to HPC.

Table 5.2 summarises the hardware characteristics of both platforms, each of them with at least one GPU-device available as an accelerator.

	Laptop Mac OS 10.9		**Compute Server** CentOS 6.3	
Device Type	**CPU**	**GPU**	**CPU**	**GPU**
Number	1	1	2	2
Manufacturer	Intel	Nvidia	Intel	Nvidia
Code	Core i7-3615	GT650M	E5-2695 v2	K20
Code Name	Ivy Bridge	GK107	Ivy Bridge	GK110
Year	2012	2012	2013	2012
Cores	4	2 MT-SIMD	2x 12	15 MT-SIMD
Core Frequency	2.3 GHz	900 MHz	2.4 GHz	705 MHz
SMT	8	2 (x192)	48	15 (x192)
Vector Support	AVX	n/a	AVX	n/a
L1 Cache	32 KiB + 32 KiB (iC+dC)	64 KiB (SMX)	32 KiB + 32 KiB (iC+dC)	64 KiB (SMX)
L2 Cache	256 KiB	256 KiB	256 KiB	1 536 KiB
L3 Cache	6 MiB (shared)	n/a	30 MiB (shared)	n/a
Main Memory	16 GiB	1 GiB	2x 32 GiB (NUMA)	4 GiB

Table 5.2.: Target hardware platforms

Performance tests were always performed by combining the CLI of *TomSeg* with automatic *scripts*, to minimise external interference and user bias. The code was compiled with GCC 4.9 (supporting OpenMP 4.0) and NVCC 7.0, with optimisation level `-O3` in both.

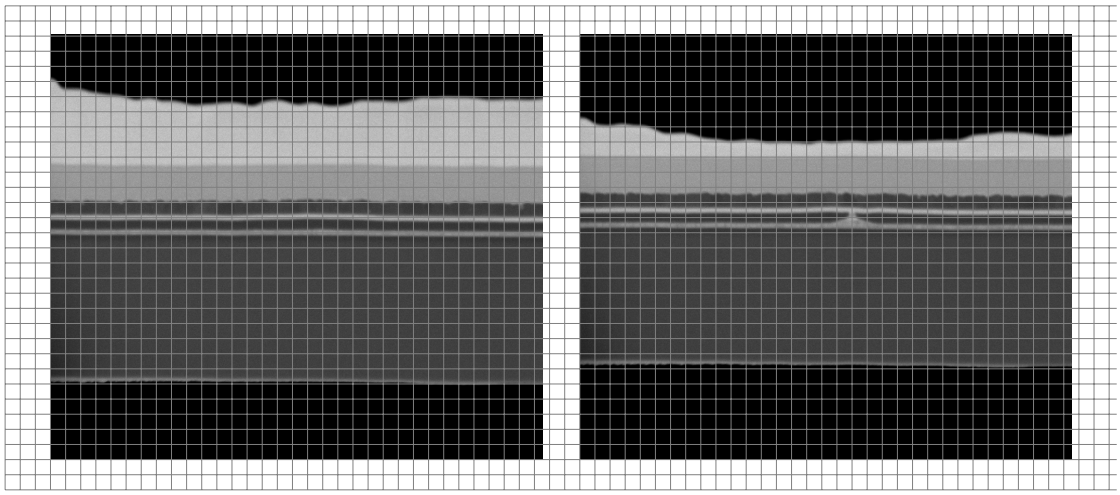
5.2 Testing the Slice Alignment

As presented in Chapter 4, *TomSeg* allows slice alignment in two distinct forms: *Align by Reference Area*, and *Align by Previous Slice*. The same chapter also refers that the former can take advantage of parallel computing directly: aligning one slice does not depend on the alignment of another. The latter method required a much more advanced approach to remove the RAW dependency: the next slice could only be aligned after the alignment of the current one.

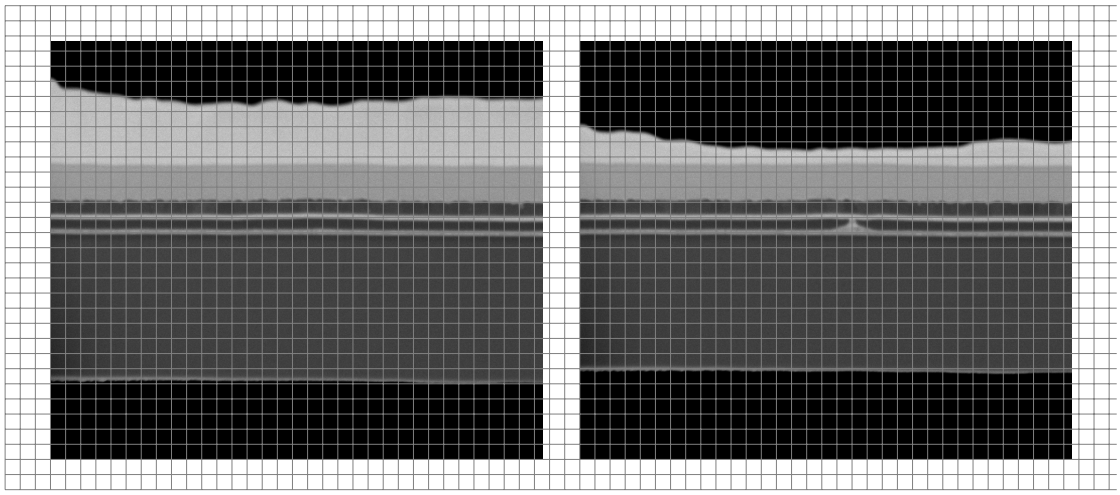
Since datasets with a same feature present along the 3rd dimension are more efficiently aligned using a *reference area*, **DS3** was validated using this approach. The other datasets were validated using an *alignment by previous slice*. By default, *TomSeg* uses the *alignment by the previous slice*, but the user can always select the other mode for the alignment.

5.2.1 *Align by Reference Area*

As already seen on the example of Figure 4.6, dataset **DS3** can be aligned using a *reference area*, common in every slices. The goal in this dataset is to align the slices by the middle, parallel, bars, which is where the main interest region of the tomogram is. Thus, that area was selected and the algorithm executed over the 131 slices. The goal was perfectly accomplished, as can be seen in Figure 5.2 with the helping grid placed over the slices 1 and 50 (randomly selected).



(a) Slices 1 & 50 misaligned



(b) Slices 1 & 50 aligned by middle bars

Figure 5.2.: Validation of the obtained alignment for DS3 (2/131 slices)

The performance results focused on analysing the speedup obtained when using multiple threads and evaluating the impact of restricting the search area to a smaller region. The alignment can be performed considerably faster if maximum values of displacement in both x and y directions are defined. The tool estimates default values as a percentage of the overall size of the image in pixels, but the user can modify them.

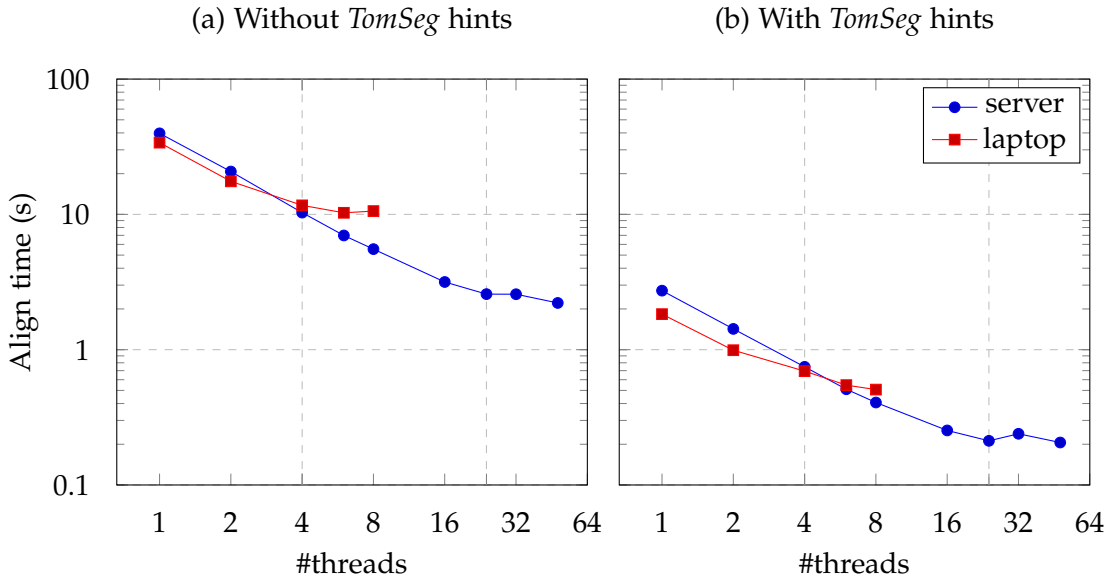


Figure 5.3.: Multithreaded performance results of aligning DS3

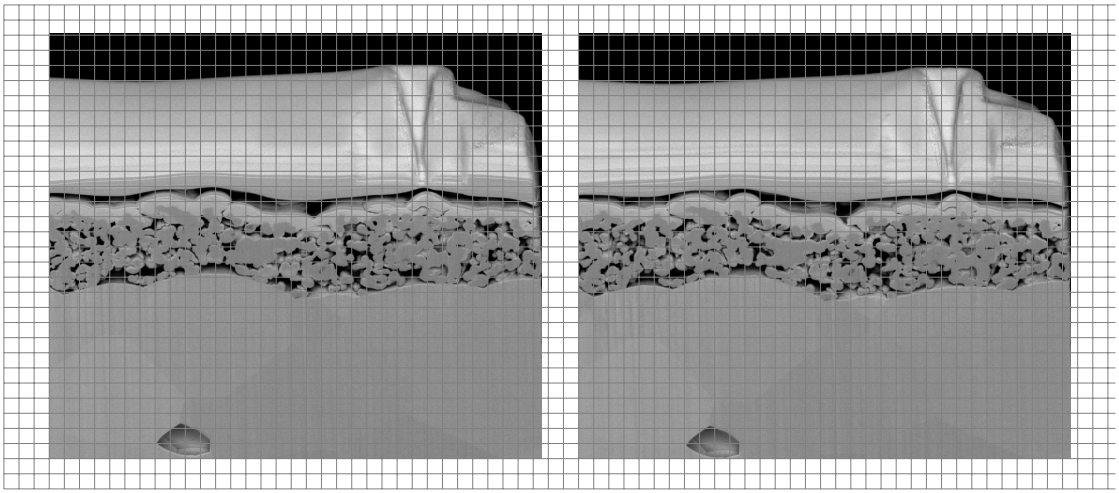
Figure 5.3 shows how multithreading helps on speeding-up slice alignment by reference areas. The execution behaviour is quite similar both on the server and on the laptop. Due to the different number of cores, they obtained maximum speedups of 13.3 and 3.6, respectively. The same figure compares the results obtained by considering or not the maximum values of slice displacements — hints. With this optimisation, and with the same number of threads, this alignment procedure of *TomSeg* ran 11 times faster.

5.2.2 *Align by Previous Slice*

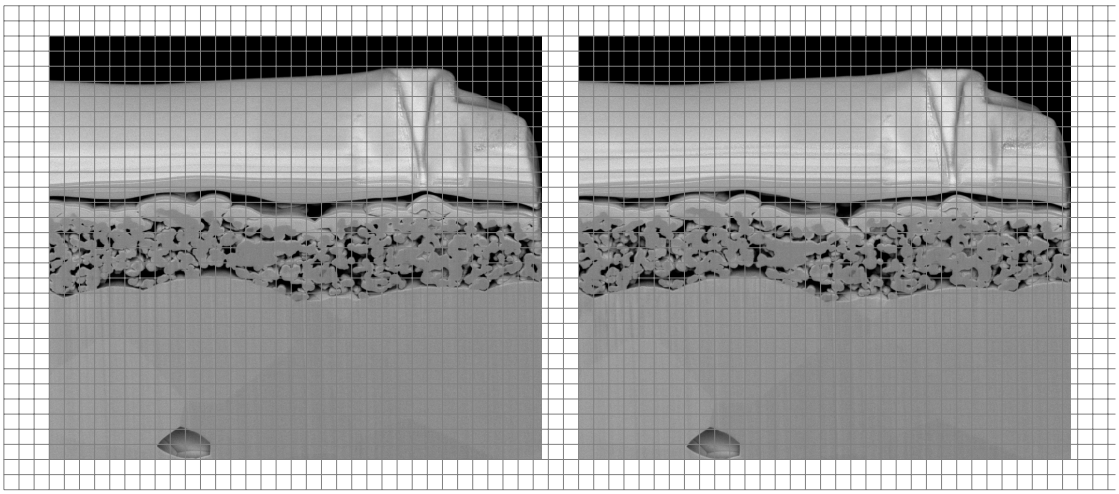
To validate this different alignment method (Figure 4.7), the datasets **DS1** and **DS2** were used. These datasets do not present any structural feature across their whole domain, thus aligning the slices by their predecessors is the best approach.

Even very hard to detect with two static images of neighbouring slices, Figures 5.4a and 5.5a show two misaligned slices from DS1 and DS2 datasets, respectively. As a result of processing the tomogram using the *alignment by previous slice* algorithm, the overall misalignment of tomogram slices was successfully corrected. Again, with the help of grid lines, Figures 5.4b and 5.5b show the resulting aligned slices.

When compared to the latter method, this is a much more computationally intensive task, as the whole image signals have to be correlated. The datasets are also larger.

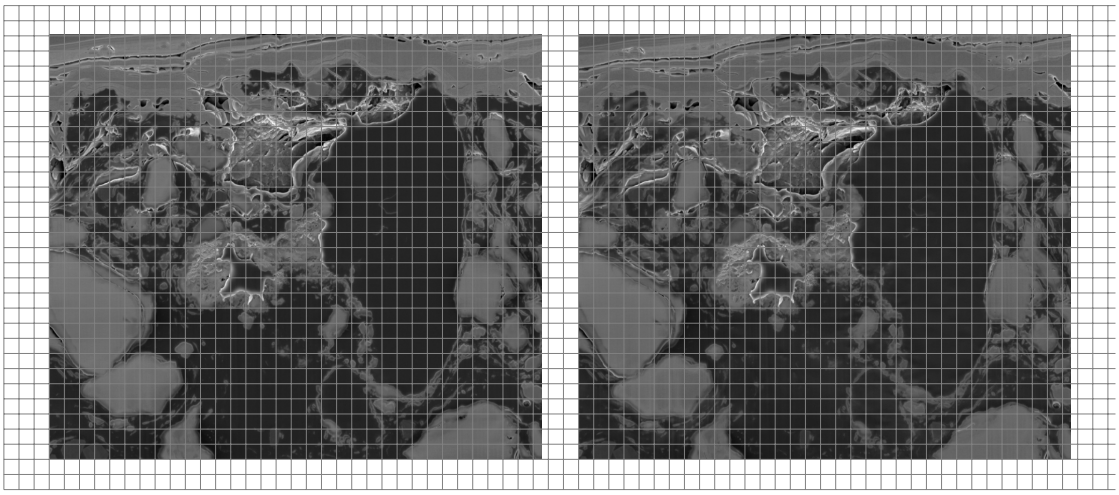


(a) Slices 1 & 2 misaligned

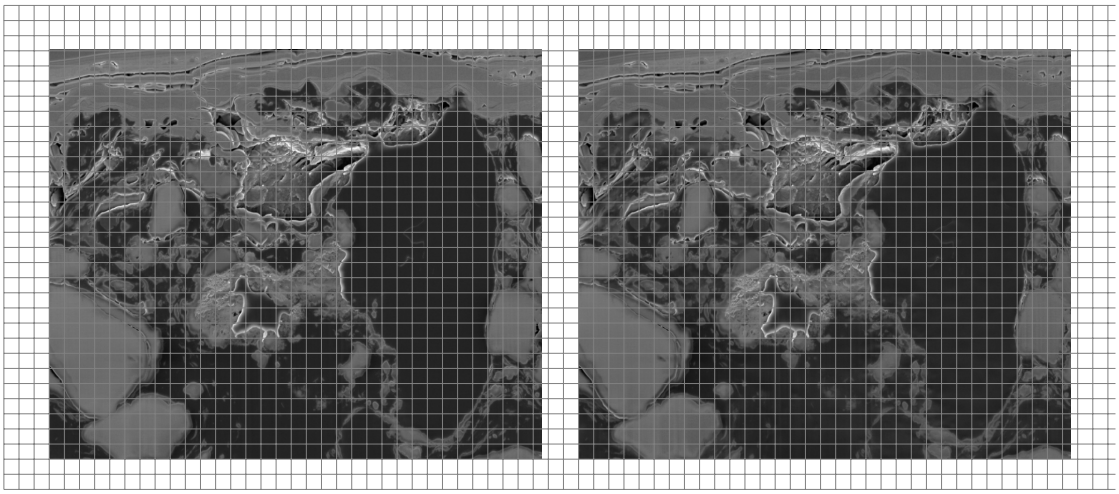


(b) Slices 1 & 2 aligned

Figure 5.4.: Validation of the obtained alignment for DS1



(a) Slices 1 & 2 misaligned



(b) Slices 1 & 2 aligned

Figure 5.5.: Validation of the obtained alignment for DS2

As expected, after eliminating the RAW dependency, the multithreaded execution of this aligning method behaves similarly to the *alignment by a reference area*.

Figure 5.6 compares directly the alignment time for the two datasets. It is interesting to note that tomograms' dimension and complexity has a huge impact on the execution time of sequential versions of the algorithm. However, as the number of threads increases, the difference between the alignment time for the two datasets becomes smaller.

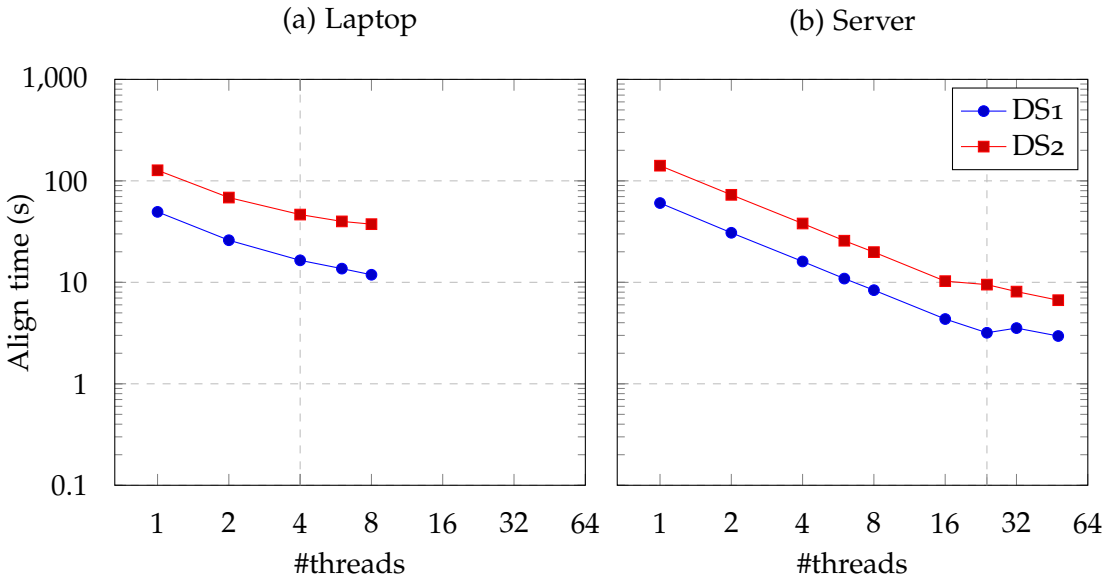


Figure 5.6.: Multithreaded performance results of aligning DS1 and DS2

When compared to the solution with the RAW dependency (sequential), this parallel one, when executed in the compute server using 24 threads, produces the exact same results, but 57 seconds faster: aligning DS1 would take 60 seconds; now it is aligned in 3. This difference is even greater when aligning bigger datasets.

5.3 TESTING THE SEEDED SLICE SEGMENTATION

The proposed process to segment tomograms follows two main phases, as seen in Section 4.5: segmenting correctly the first slice of a sub-tomogram — a *seeded slice* — using *Proportional Region Growing*; and computing the segmentation of the following slices based on the method *Difference Between Slices*.

To validate the former phase *seeded slices* from all datasets were tested. Table 5.3 presents the obtained segmentation results. The *seeded slices* from the datasets **DS2** and **DS3** were successfully segmented. However, the segmentation of **DS1** is not as detailed as desired: when a single slice contains information about particles of multiple depths, which typically occurs in porous materials, *TomSeg* v1.0 does not recognise them as background *voxels*.

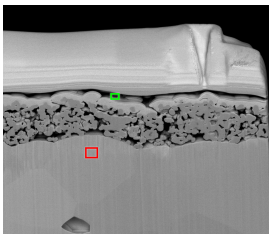
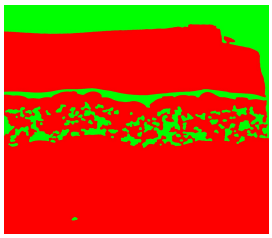
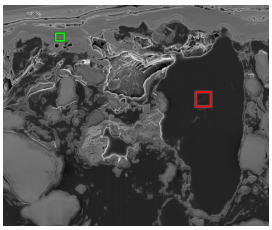
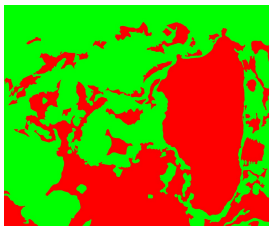
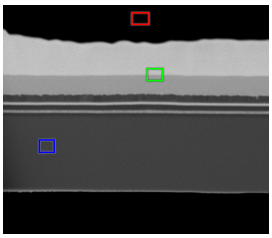
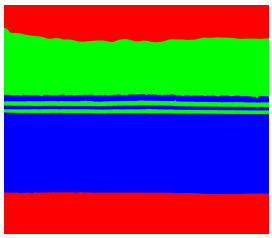
Dataset	Seeded Slice	Seg. Result	Mininum Feature Size
DS1			15px
DS2			20px
DS3			15px

Table 5.3.: Validation of the segmentation results of *seeded slices*

With the topographic effects experienced in porous materials, a continuity of grey-levels between material and pore *voxels* is visible. This makes automatic segmentation of these kind of reconstructed tomograms very hard, or even impossible. *TomSeg* is not an exception: with methods based on the analysis of histogram data (*Initial Conquer* and *Automatic Conquer*) at its core and with no mechanism to detect the existence of pores in the tomogram, background voxels (represented with attenuated intensities) are often interpreted as being material.

The algorithm to segment *seeded slices* was left essentially sequential, since parallelism is achieved through the segmentation of multiple sub-tomograms. To understand how processing time was being distributed by the different phases, the execution was analysed with the `callgrind` tool. By recurring to the generated call-graphs, the identification of main bottlenecks became trivial. Tests were also performed to evaluate the impact of offloading some tasks to GPU-devices, or even creating nested parallelism. Results are reported ahead.

Figure 5.7 presents a summarised call-graph obtained for the first version of *Proportional Region Growing*, segmenting one random *seeded slice* from the **DS3** dataset.

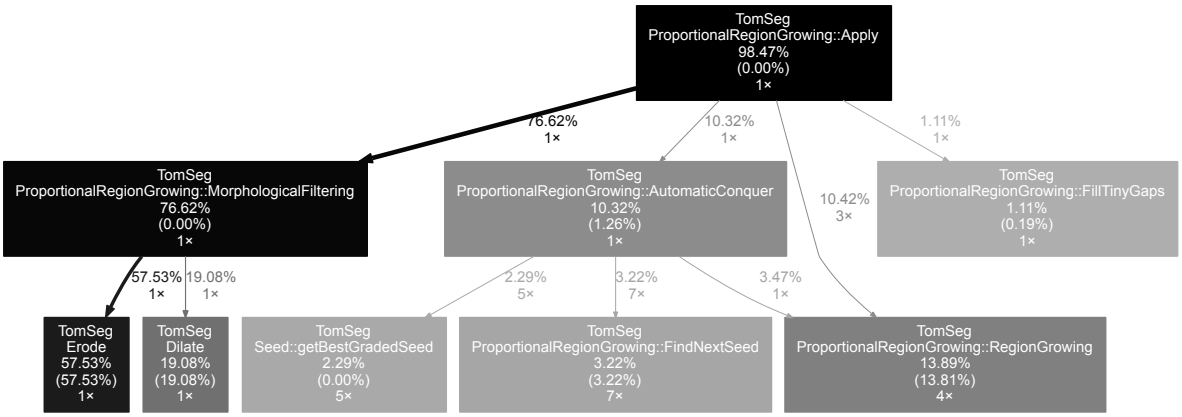


Figure 5.7.: Call-graph of the first version of *Propor. Region Growing* (DS3)

As can be seen, the `MorphologicalFiltering` phase represented 77% of the overall slice segmentation time. Any improvement on this phase will have a significative impact on the overall segmentation time. To improve it, several alternatives were targeted:

- adapt the solution to support OpenCV's vectorised implementation;
- adapt the solution to support OpenCV's GPU implementation;
- develop a new custom parallel version to offload the task to a GPU-device.

Figure 5.8 shows the measured execution times of each alternative to *Morphological Filtering* on the compute server. All solutions led to much more satisfactory results. From these, and to fit in the characteristics of the specific platforms, *TomSeg* allows users to opt between the alternative based on the morphological filters of OpenCV (CPU mode); and the custom parallel version targeting an accelerator device (GPU mode). With the OpenCV optimisation (CPU mode), the overall segmentation of DS3 had a speedup of 3.

After this change, the method `RegionGrowing` became the main bottleneck, taking more than 57% of the segmentation time. This method is a high order function, called multiple times during both *Initial* and *Automatic Conquer* steps, as detailed in Section 4.5.3. Even without accessing *voxels* from another slices, a preliminary profile using the hardware performance counters pointed to problems on accessing slice data. In fact, after a deeper

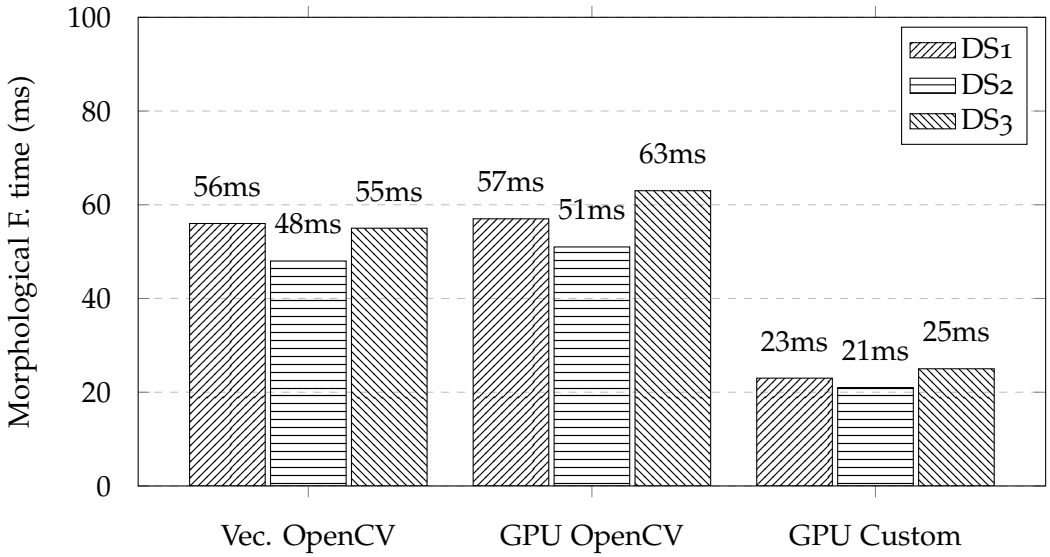


Figure 5.8.: Performance results of versions of *Morphological Filtering*

code analysis, it was concluded that the pattern in which the neighbouring elements were being inspected was not cache-friendly. With the new version of the algorithm visiting the neighbouring elements row-wisely, the number of *cache* misses dropped considerably. Additionally, the new algorithm supports more vectorised operations and the number of recursive calls was reduced. Generally, this review led to a sequential speedup of 2.8 for DS3 (on the server).

Further performance improvements were introduced, namely searching for new automatic seeds from the last searched point. The first approach starts the search from the beginning of the partial result. However, since the *region growing* process is performed right after finding a new seed, at the end of this process it is known that the region above the seed is already segmented. That allows the search for new automatic seeds to start from the last found one.

Even after this improvement, the `AutomaticConquer` phase continued to be a significant execution bottleneck (45%): whenever a new seed is found, *TomSeg* needs to determine which class it represents. To do such classification several metrics are computed and analysed. The most compute demeaning one is to determine the distance between the new seed and the closest (already) segmented *voxel* of each class. As part of a complex grading formula, this metric may not be decisive for making a decision. With this last optimisation,

			DS1 (s)	DS2 (s)	DS3 (s)
		`ProportionalRegionGrowing` (v0.10)	7.68	7.46	5.15
v0.10	+	`MorphologicalFiltering` using OpenCV (v0.11)	4.37	4.45	1.53
v0.11	+	Vect & Cache-friendly `RegionGrowing` alg. (v0.12)	3.58	4.18	0.54
v0.12	+	`FindNewSeeds` from last searched point (v0.13)	3.25	3.72	0.46
v0.13	+	`GetBestGradedSeed` reordered (v0.14)	0.34	0.39	0.17

Table 5.4.: Progressive optimisations and obtained results

the distance metric is only computed when it is strictly needed. This small adjustment had a gigantic repercussion. For the datasets in which the *Automatic Conquer* phase is more expressive, the overall segmentation became up to 9 times faster (DS2, on the server).

Table 5.4 summarises the optimisations introduced, presenting their respective impact on the overall segmentation time of each dataset; only one *seeded slice* from each dataset was considered. When compared to the first version, segmenting DS3 became 30 times faster.

Figure 5.9 shows the final distribution of the execution time by the different phases of *Proportional Region Growing*, using the previous dataset — DS3. It is interesting to note that, with the introduced improvements, the execution time of the `Apply` method dropped from 98% to 37% of the global execution time of *TomSeg*.

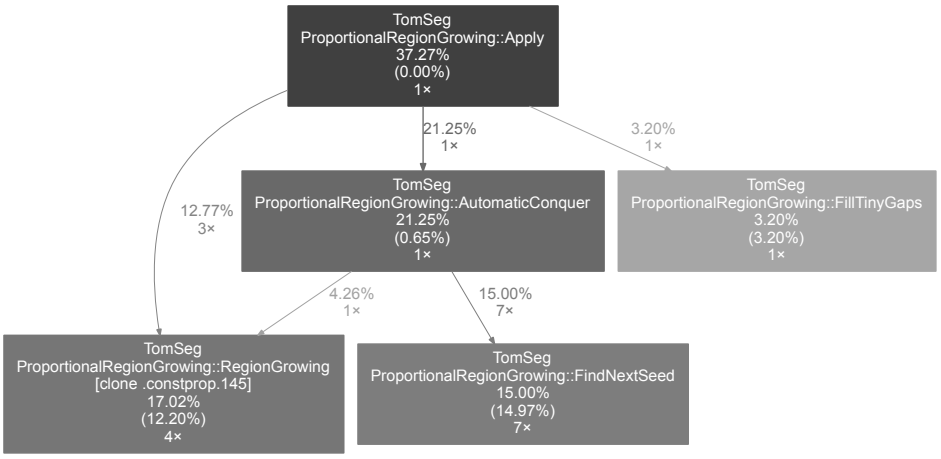


Figure 5.9.: Call-graph from the last version of *Propor. Region Growing* (DS3)

This figure also confirms that `MorphologicalFiltering` is no longer a performance issue. Its impact was so small that it was not even represented on the new call-graph.

5.4 TESTING THE TOMOGRAM SEGMENTATION

The next and last step is to test the segmentation on the 3D space. Since this is one of the main goals of this work, a more detailed analysis is performed to each dataset, starting with the most interesting dataset to test, **DS3**.

Validation of 3D Segmentation on DS3

The dataset **DS3** was used to validate the capacity of *TomSeg* in handling a tomogram across each of its steps. It requires the combination of all functionalities: from slice alignment to exporting the results to a visualisation tool. DS3 dataset presents deeply misaligned slices and its feature of interest is only a tiny portion of the volume. The results obtained on the alignment step are shown in Figure 5.2. After aligned, the volume was cropped, forming a new volume of 91 slices, each with 459×108 *voxels* (Figure 5.10).

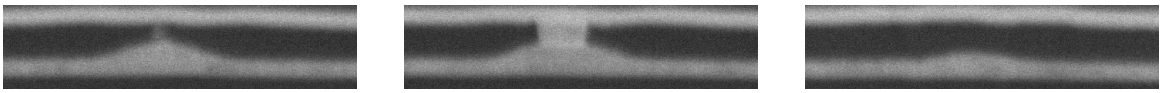


Figure 5.10.: Random slices from $DS3_{crop}$ (3/91)

With 91 slices to segment, seeds were defined on the first slice and propagated with a stride of 7 to the others, forming 13 independent sub-volumes. As also shown on Section 4.7.2, Figure 5.11 shows the obtained results of segmentation in the 3D space. The 3D visualisation was performed using *Chimera* with no smoothing, allowing a fair validation of the results.

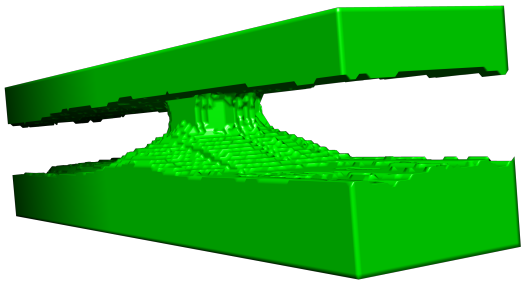
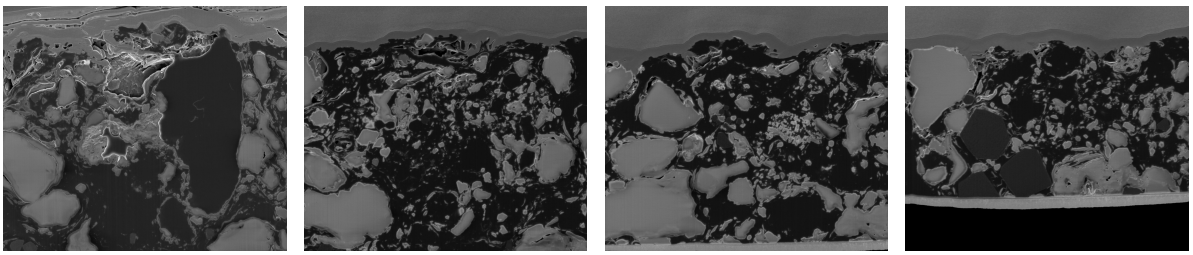


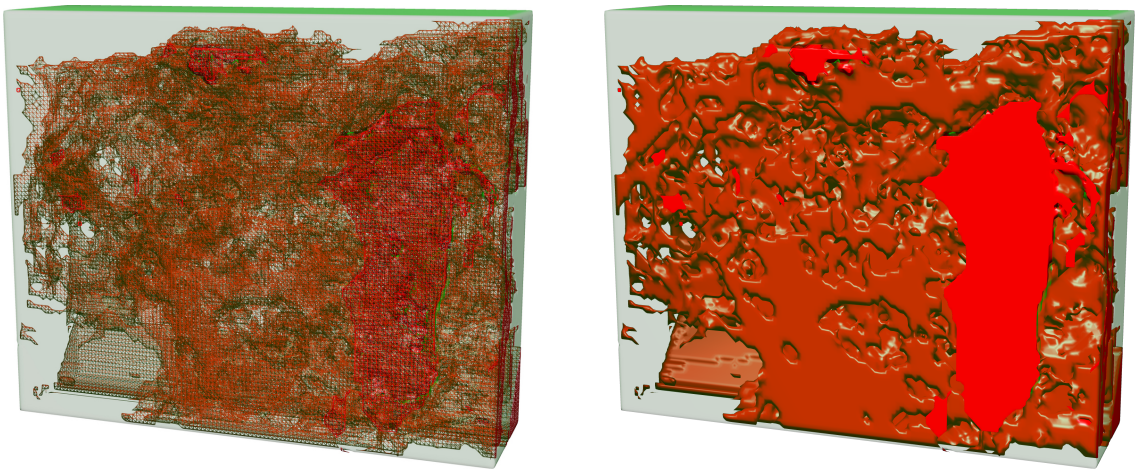
Figure 5.11.: 3D Segmentation results of $DS3_{crop}$

Validation of 3D Segmentation on DS2

The **DS2** dataset is the larger and the more complex from the three datasets. Some slices from the beginning of the volume must be cropped. They present high levels of noise and do not correspond to actual slices of the volume. Figure 5.12 shows the details of the volume in this stack of images, and two different views over the segmented volume: on the left a mesh view, that makes easier to visualise the deepness of the tomogram; on the right the surface view.



(a) Slice 82 (b) Slice 150 (c) Slice 300 (d) Slice 600



(e) Mesh view (f) Surface view

Figure 5.12.: 3D Segmentation of DS2

The obtained results are very satisfactory and detailed, confirming the capacity of *TomSeg* to segment complex unstructured volumes, as long as their slices do not present discontinuities. As can be denoted, small details can be easily identified, which is only possible due to the high resolution of the segmentation result (detailed view available in Figure 5.13).[1]

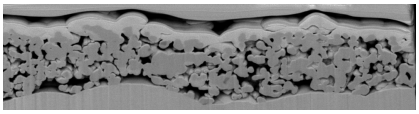
1 *Minimum Feature Size* was defined to be 20px.

Figure 5.13.: Detailed view of DS2 3D Segmentation result

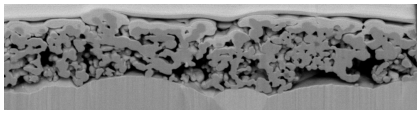
Validation of 3D Segmentation on a cropped VOI of DS1

The original **DS1** tomogram contains irrelevant information that can be discarded after aligning the images. This, besides improving the *TomSeg* performance and helping on the analysis, can also represent a fundamental step to obtain valid results.

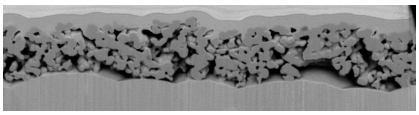
Figure 5.14 shows the slices and the obtained segmentation results after cropping the volume, which only kept the relevant information: the middle porous area (details of VOI selection in Section 4.4).



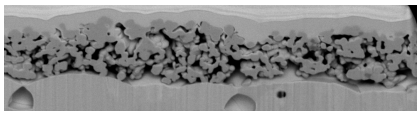
(a) Slice $1_{\text{crop.}}$



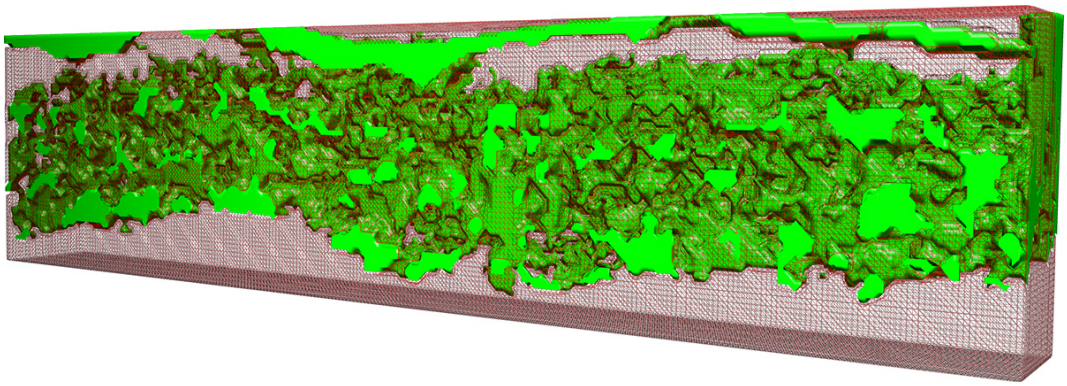
(b) Slice $50_{\text{crop.}}$



(c) Slice $150_{\text{crop.}}$



(d) Slice $203_{\text{crop.}}$



(e) Obtained results visualised in *Chimera* (red: mesh; green: surface)

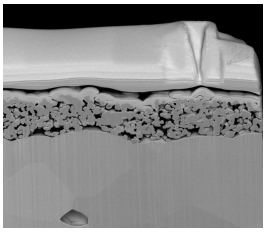
Figure 5.14.: 3D Segmentation of DS1_{crop}

Although not perfect, the obtained results are very satisfactory to perform a qualitative study on the sample. Even with low resolution results, the pores (green class) can be distinguished from the material (red class) without any problem, making qualitative analysis possible.

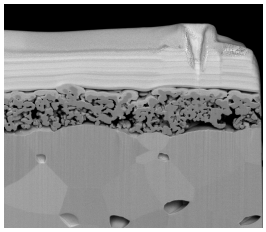
To obtain segmentation results with higher resolution it is fundamental to decrease/detect the *voxels* which depict more than one depth.

Validation of 3D Segmentation on a full view of DS1

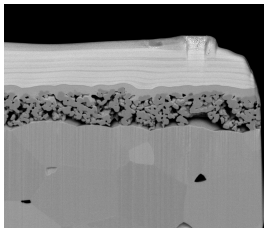
As detailed on Section 5.3, the current version of *TomSeg* has some limitations while segmenting tomograms whose slices depict more than one depth of the sample. That is the case of **DS1**. To understand and clarify this limitation Figure 5.15 shows the obtained segmentation result for the full view of DS1, using the current version of *TomSeg* (v1.0).



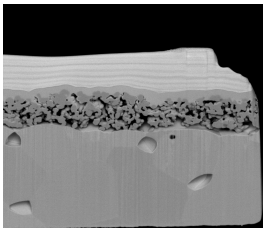
(a) Slice 1



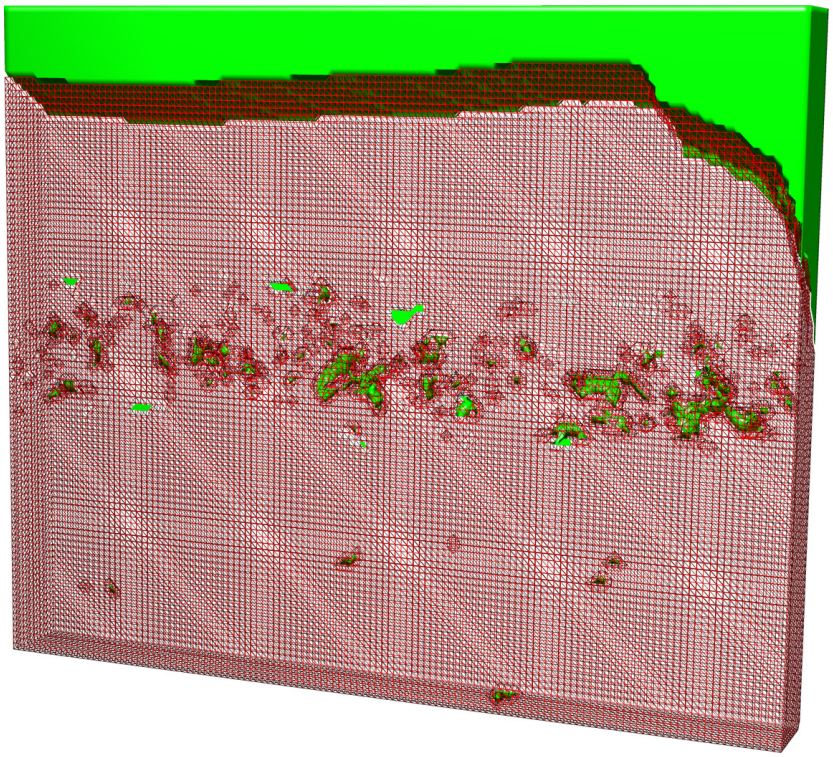
(b) Slice 50



(c) Slice 150



(d) Slice 203



(e) Obtained results visualised in *Chimera* (red: mesh; green: surface)

Figure 5.15.: 3D Segmentation of DS1

Unlike the other datasets, the obtained segmentation result considering the full view of DS1 is not acceptable. Figure 5.15e presents imperfections on the porous material (green

phase): only the larger portions are segmented with the expected resolution. On the smaller regions, the *Automatic Conquer* algorithm takes incorrect decisions on which class the new automatic seeds belong: it can not distinguish between the pores and the material just by inspecting the *voxels* intensities. Comparing to the results obtained for the cropped VOI version of DS1 (Figure 5.14), it becomes clear that in this kind of tomograms the resolution decreases as the tomogram complexity increases: the less complex the tomograms are, the fewer errors *TomSeg* will introduce by miss-segmenting background *voxels*.

Performance Analysis of 3D Segmentation

To evaluate the performance, results are displayed and discussed relative to the full **DS3**. Figure 5.16 shows the 3D Segmentation times measured using different configurations of DS3, i.e., the segmentation of the same tomogram was performed with different numbers of *seeded slices*. As seen before, the volume can be divided in independent sub-volumes at each *seeded slice*. This analysis tries to find the best balance between the parallelism degree and the number of slices which need to be fully processed with `ProportinalRegionGrowing`.

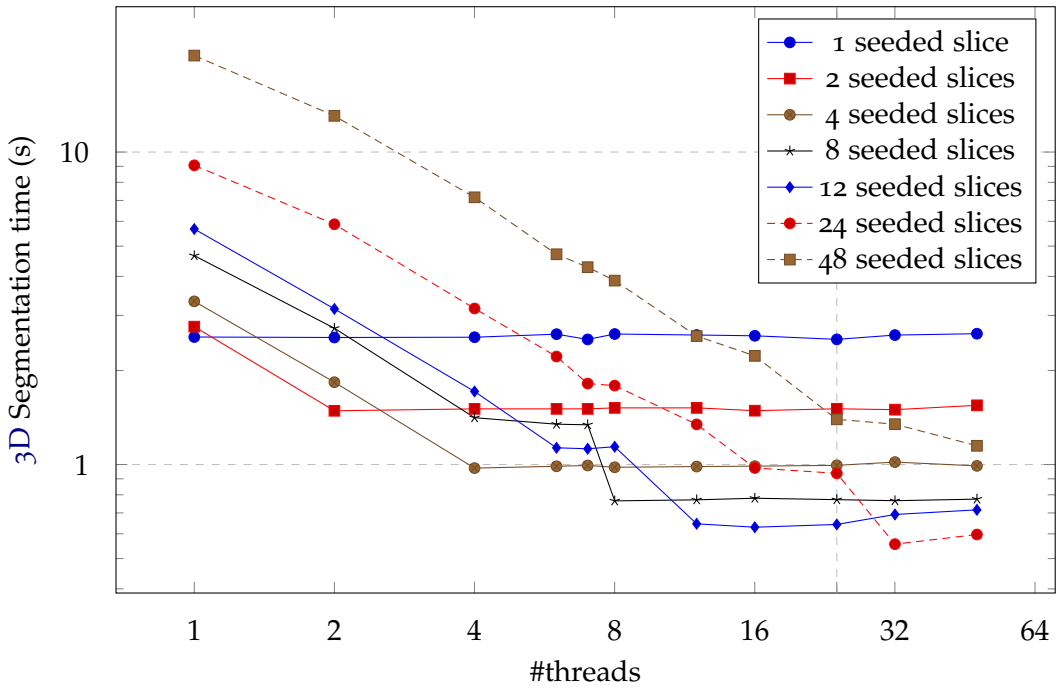


Figure 5.16.: Performance analysis using multiple configurations (DS3, server)

As expected, when several *seeded slices* are available but parallelism is not, the performance is degraded. However, the segmentation becomes faster as the parallelism degree increases. On the other hand, when only one *seeded slice* is defined — no independent sub-volumes can be created — the execution time remains constant. The same is seen when the number of threads exceeds the number of sub-volumes/*seeded slices*.

Even with hardware support to process each independent sub-tomogram in parallel, the obtained results using 48 *seeded slices* is not the best. This is mainly due to the incapacity of processor's simultaneous multithreading technology to hide the extra time needed to fully segment the additional *seeded slices*.

The best results were obtained using 24 *seeded slices* (forming 24 independent groups of 5 slices), representing a speedup of 15x when compared to segmenting the volume sequentially. It is interesting to note that the best results were obtained when the number of independent sub-tomograms is equal to the number of physical cores of the processor. Using this configuration, segmenting the 131 slices only took 0.55 seconds.

6

CONCLUSIONS

This dissertation presented a novel approach to segment 3D tomograms that combines concepts of efficient computing with the needs of ET experiments. The result was the development of a new algorithm that requires minimal user intervention to efficiently segment Slice and View tomograms. The implementation of this algorithm is packed in a new software tool, *TomSeg*, developed to offer the best experience while segmenting tomograms. Different pre-processing tools were also included to free the user from using other software packages to perform additional actions, such as slice alignment and VOI definition.

The developed solution targets tomograms obtained with ET Slice and View techniques. The validation of the results was performed recurring to datasets obtained using FIB-SEM serial sectioning, which allowed to test each component of *TomSeg*. With the exception of segmenting porous materials, where several depths are depicted into the slices, the obtained results showed that this new approach is very satisfactory to perfectly align slices and to segment non structured materials, using the available computational resources properly.

The modular design of *TomSeg* was crucial to facilitate an agile development. Its two distinct user interfaces permitted to test and tune the solution's performance in different execution environments. This allowed for *TomSeg* to be easily tested and validated on a personal computer and on two compute servers, one of them based on a 64-core *Intel's Xeon Phi Knights Landing*.

The challenges presented in Chapter 1, which raised the problems of handling tomogram data, were overcome for Slice and View tomograms: this dissertation work showed that the development of efficient algorithms to handle tomogram data requiring minimal user intervention is viable. The modular features of *TomSeg*, complemented with advanced

functionalities, make it a software package in where new solutions and ideas can be easily developed and validated.

6.1 FUTURE WORK

The developed tool and the obtained results motivate further research on tomogram handling and, in particular, region growing algorithms. Some of the most interesting possibilities open to future improvements are:

- Validate and adjust the solution using tomograms of different sample types (e.g., biological specimens); obtained using different EM techniques (e.g., TEM-Tilt series); obtained using different imaging methods (e.g., X-ray);
- Extend *TomSeg* algorithms to segment tomograms not only considering density maps obtained using imaging techniques, but also considering data collected through other sensors available in the microscope;
- Design and integrate a new phase to 3D Segmentation process capable of identifying slice regions that do not represent its actual depth;
- Port the `RegionGrowing` algorithm to CUDA, extending the current support for GPUs;
- Develop a module to compute quantitative metrics (e.g., *voxels* per class, surface area);
- Take advantage of multiple compute servers to accelerate segmentation: `tms` project files, allied to a distributed file system, can be used to scatter the sub-volumes;
- Develop (or import) a real-time 3D visualisation tool;
- Develop a web-interface to generate `tms` project files interactively from the browser[1];
- Develop a `TomSegManager` web-server;
- Check the feasibility of using neural networks to classify the *degenerated seeds*;
- Check the feasibility of using 3D region growing algorithms.

1 These files are JSON based

BIBLIOGRAPHY

Rolf Adams and Leanne Bischof. Seeded region growing. *IEEE Transactions on pattern analysis and machine intelligence*, 16(6):641–647, 1994.

Serge Beucher and Fernand Meyer. The morphological approach to segmentation: the watershed transformation. *Optical Engineering*, 34:433–433, 1992.

Ralph A Bradshaw and Philip D Stahl. *Encyclopedia of Cell Biology*. Academic Press, 2015.

Anchi Cheng, Richard Henderson, David Mastronarde, Steven J Ludtke, Remco HM Schoenmakers, Judith Short, Roberto Marabini, Sargis Dallakyan, David Agard, and Martyn Winn. Mrc2014: Extensions to the mrc format header for electron cryo-microscopy and tomography. *Journal of structural biology*, 2015.

Tony J Collins. Imagej for microscopy. *BioTechniques*, 43(1 Suppl):25–30, 2007.

Antony R Crowther, Richard Henderson, and John M Smith. Mrc image processing programs. *Journal of structural biology*, 116(1):9–16, 1996.

Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.

Winfried Denk and Heinz Horstmann. Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure. *PLoS Biol*, 2(11):e329, 2004.

Joachim Frank. *Electron Tomography: Three-Dimensional Imaging with the Transmission Electron Microscope*. Springer US, 1992.

Joachim Frank. *Electron tomography: methods for three-dimensional visualization of structures in the cell*. Springer Science & Business Media, 2008.

Edgar Garduño, Mona Wong-Barnum, Niels Volkmann, and Mark H Ellisman. Segmentation of electron tomographic data sets using fuzzy set theory principles. *Journal of structural biology*, 162(3):368–379, 2008.

William Gropp, Ewing Lusk, and Rajeev Thakur. *Using MPI-2: Advanced features of the message-passing interface*. MIT press, 1999.

Jochen Joos, Thomas Carraro, André Weber, and Ellen Ivers-Tiffée. Reconstruction of porous electrodes by fib/sem for detailed microstructure modeling. *Journal of Power Sources*, 196(17):7302–7307, 2011.

James R Kremer, David N Mastronarde, and Richard McIntosh. Computer visualization of three-dimensional image data using imod. *Journal of structural biology*, 116(1):71–76, 1996.

Robert M Lewitt. Reconstruction algorithms: Transform methods. *Proceedings of the IEEE*, 71(3):390–408, 1983.

Nihar Mahapatra and Balakrishna Venkatrao. The processor-memory bottleneck: Problems and solutions. *Crossroads*, 5(3es), 1999.

Richard McIntosh, Daniela Nicastro, and David Mastronarde. New views of cells in 3d: an introduction to electron tomography. *Trends in cell biology*, 15(1):43–51, 2005.

Rajesh Narasimha, Iman Aganj, Adam E Bennett, Mario J Borgnia, Daniel Zabransky, Guillermo Sapiro, Steven W McLaughlin, Jacqueline LS Milne, and Sriram Subramaniam. Evaluation of denoising algorithms for biological electron tomography. *Journal of structural biology*, 164(1):7–17, 2008.

Nvidia. Nvidias next generation cuda compute architecture: Kepler gk110. *Whitepaper*, 2012.

Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11 (285-296):23–27, 1975.

Nikhil R Pal and Sankar K Pal. A review on image segmentation techniques. *Pattern recognition*, 26(9):1277–1294, 1993.

David A Patterson and John L Hennessy. *Computer organization and design: the hardware/software interface*. Morgan Kaufmann, 5th edition, 2013.

Eric F Pettersen, Thomas D Goddard, Conrad C Huang, Gregory S Couch, Daniel M Greenblatt, Elaine C Meng, and Thomas E Ferrin. Ucsf chimera — a visualization system for exploratory research and analysis. *Journal of computational chemistry*, 25(13):1605–1612, 2004.

Raihana Ruhaiyem. Semi-automated cellular tomogram segmentation workflow (ctsw): Towards an automatic target-scoring system. *The International Conference on Computer Graphics, Multimedia and Image Processing*, 2014.

Richard M Russell. The cray-1 computer system. *Communications of the ACM*, 21(1):63–72, 1978.

Shane Ryoo, Christopher I Rodrigues, Sara S Baghsorkhi, Sam S Stone, David B Kirk, and Wen-mei W Hwu. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 73–82. ACM, 2008.

Detlev Stalling, Malte Westerhoff, and Hans-Christian Hege. Amira: A highly interactive system for visual data analysis. *The visualization handbook*, 38:749–67, 2005.

Wei T Tsai, Ahmed Hassan, Purbasha Sarkar, Joaquin Correa, Zoltan Metlagel, Danielle M Jorgens, and Manfred Auer. From voxels to knowledge: a practical guide to the segmentation of complex electron microscopy 3D-data. *Journal of Visualized Experiments*, 2014.

Niels Volkmann. A novel three-dimensional variant of the watershed transform for segmentation of electron density maps. *Journal of structural biology*, 138(1):123–129, 2002.

Dongguang Wei, Scott Jacobs, Shannon Modla, Shuang Zhang, Carissa Young, Robert Cirino, Jeffrey Caplan, and Kirk Czymmek. High-resolution three-dimensional reconstruction of a whole yeast cell using focused-ion beam scanning electron microscopy. *BioTechniques*, 53(1):41–48, 2012.

A

THE MRC FILE FORMAT

The MRC format is widely used in the 3D EM field due to its simplicity and efficiency (Crowther et al., 1996). It was created at the Medical Research Council Laboratory of Molecular Biology in the 1980s to represent images and volumes, and continues to be used in a large suite of programs.

The format describes a binary file with three main sections. The first — the main header — contains fixed format values for metadata about the volume. The main header is limited to 1024 bytes, but includes unassigned space allowing custom extensions. The second section is a variable length extended header. Finally, the third section contains the actual volume data with grid values represented as one of a range of possible data types according to the *mode* of the map (Cheng et al., 2015).

The file body organises the different volume slices sequentially in memory, turning the process of reading and writing a file to and from the main memory straightforward.

Being a generic format for 3D images, some of its header fields are not relevant in the context of this work. Table A.1 summarises the relevant fields, and the Figure A.1 presents the complete MRC format (with those fields also highlighted).

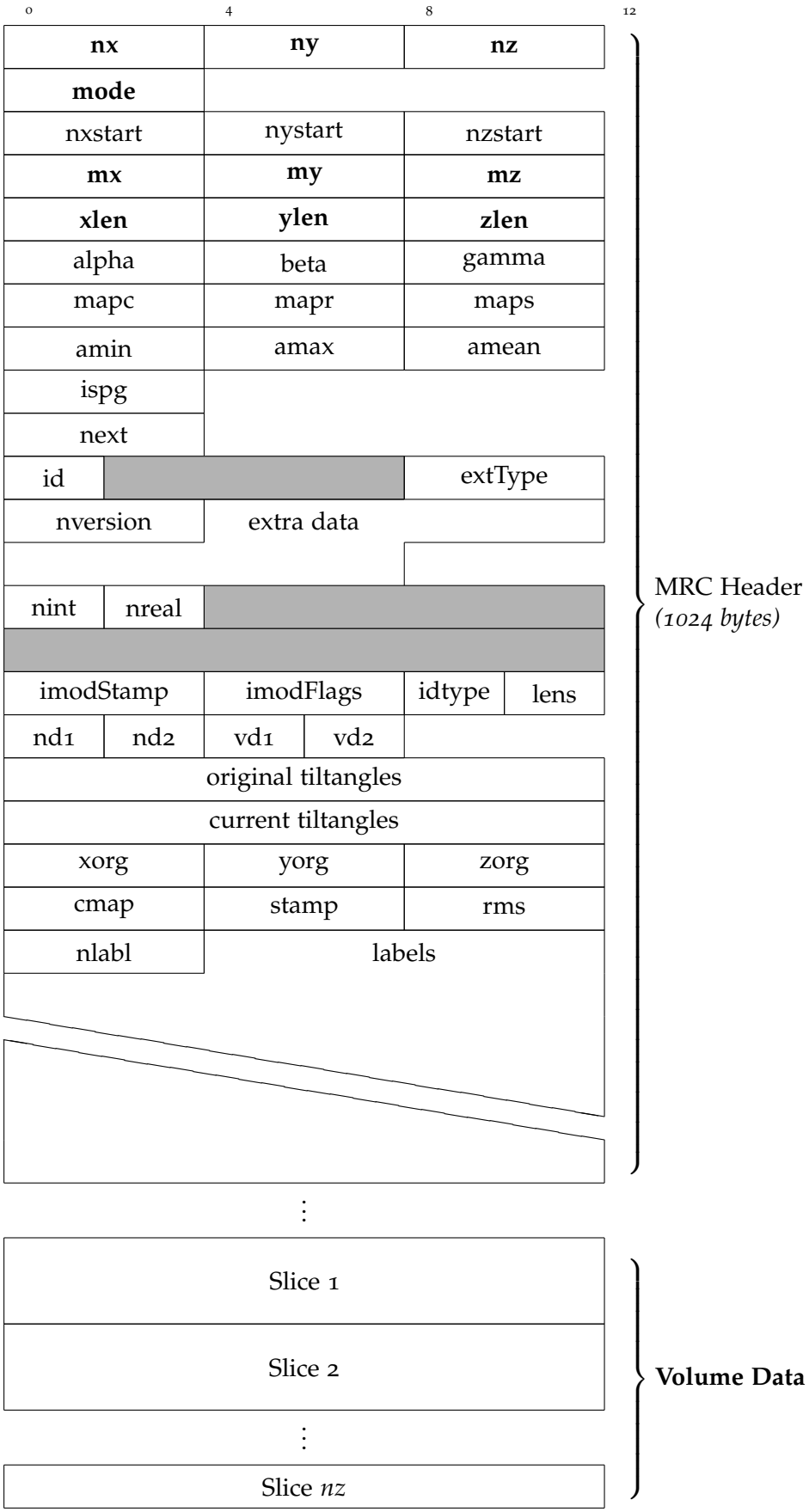


Figure A.1.: MRC file format scheme (names of C code; sizes in bytes)

Name	Description
nx	Number of Columns
ny	Number of Rows
nz	Number of Sections
mode	Type of Pixels (0 = bytes)
mx	Grid size in X
my	Grid size in Y
mz	Grid size in Z
xlen	Cell size in X
ylen	Cell size in Y
zlen	Cell size in Z

Table A.1.: Description of the relevant header fields of the MRC file format